CHAPTER 4

Pattern Recognition

Spoken language processing relies heavily on pattern recognition, one of the most challenging problems for machines. In a broader sense, the ability to recognize patterns forms the core of our intelligence. If we can incorporate the ability to reliably recognize patterns in our work and life, we can make machines much easier to use. The process of human pattern recognition is not well understood.

Due to the inherent variability of spoken language patterns, we emphasize the use of statistical approach in this book. The decision for pattern recognition is based on appropriate probabilistic models of the patterns. This Chapter presents several mathematical fundamentals for statistical pattern recognition and classification. In particular, Bayes decision theory, and estimation techniques for parameters of classifiers are introduced. Bayes decision theory, which plays a central role for statistical pattern recognition, is described to introduce the concept of decision-making based on both *posterior* knowledge obtained from specific observation data, and *prior* knowledge of the categories. To build such a classifier or predictor, it is critical to estimate *prior* class probabilities and the class-conditional probabilities for a Bayes classifier.

Supervised learning has class information for the data. Only the probabilistic structure needs to be learned. Maximum likelihood estimation (MLE) and maximum posterior probability estimation (MAP) that we discussed in Chapter 3 are two most powerful methods. Both MLE and MAP aim to maximize the likelihood function. The MLE criterion does not necessarily minimize the recognition error rate. Various discriminant estimation methods are introduced for that purpose. *Maximum mutual information estimation* (MMIE) is based on criteria to achieve maximum model separation (the model for the correct class is well separated from other competing models) instead of likelihood criteria. The MMIE criterion is one step closer but still is not directly related to minimizing the error rate. Other discriminant estimation methods, such as *minimum error-rate estimation*, use the ultimate goal of pattern recognition – minimizing the classification errors. *Neural networks* are one class of discriminant estimation methods.

The EM algorithm is an iterative algorithm for unsupervised learning in which class information is unavailable or only partially available. The EM algorithm forms the theoretical basis for training hidden Markov models (HMM) as described in Chapter 8. To better understand the relationship between MLE and EM algorithms, we first introduce *vector quantization* (VQ), a widely used source-coding technique in speech analysis. The well-known *k-means* clustering algorithm best illustrates the relationship between MLE and the EM algorithm. We close this chapter by introducing a powerful binary prediction and regression technique, classification and regression trees (CART). The CART represents an important technique that combines rule-based expert knowledge and statistical learning.

4.1. BAYES DECISION THEORY

Bayes decision theory forms the basis of statistical pattern recognition. The theory is based on the assumption that the decision problem can be specified in probabilistic terms and that all of the relevant probability values are known. Bayes decision theory can be viewed as a formalization of a common-sense procedure, i.e., the aim to achieve minimum-error-rate classification. This common-sense procedure can be best observed in the following real-world decision examples.

Consider the problem of making predictions for the stock market. We use the Dow Jones Industrial average index to formulate our example, where we have to decide tomorrow's Dow Jones Industrial average index in one of the three categories (events): *Up*, *Down*, or *Unchanged*. The available information is the probability function $P(\omega)$ of the three categories. The variable ω is a discrete random variable with the value $\omega = \omega_i \ (i = 1, 2, 3)$. We call the probability $P(\omega_i)$ a *prior* probability, since it reflects prior knowledge of tomorrow's Dow Jones Industrial index. If we have to make a decision based only on the *prior* probability, the most plausible decision may be made by selecting the class ω_i with the highest prior probability $P(\omega_i)$. This decision is unreasonable, in that we always make the same decision even though we know that all three categories of Dow Jones Industrial index changes will appear. If we are given further observable data, such as the federal-funds interest rate or the jobless rate, we can make a more informed decision. Let x be a continuous random variable whose value is the federal-fund interest rate, and $f_{x|\omega}(x \mid \omega)$ be a class-conditional pdf. For simplicity, we denote the pdf $f_{x|\omega}(x \mid \omega)$ as $p(x \mid \omega_i)$, where *i* = 1, 2, 3 unless there is ambiguity. The class-conditional probability density function is often referred to as the *likelihood* function as well, since it measures how likely it is that the underlying parametric model of class ω_i will generate the data sample *x*. Since we know the prior probability $P(\omega_i)$ and class-conditional pdf $p(x \mid \omega_i)$, we can compute the conditional probability $P(\omega_i \mid x)$ using Bayes' rule:

$$P(\omega_i \mid x) = \frac{p(x \mid \omega_i) P(\omega_i)}{p(x)} \tag{4.1}$$

where $p(x) = \sum_{i=1}^{3} p(x \mid \omega_i) P(\omega_i)$.

The probability term in the left-hand side of Eq. (4.1) is called the *posterior* probability as it is the probability of class ω_i after observing the federal-funds interest rate *x*. An intuitive decision rule would be choosing the class ω_k with the greatest posterior probability. That is,

$$k = \arg\max_i P(\omega_i \mid x) \tag{4.2}$$

In general, the denominator $p(x)$ in Eq. (4.1) is unnecessary because it is a constant term for all classes. Therefore, Eq. (4.2) becomes

$$k = \arg\max_i P(\omega_i \mid x) = \arg\max_i p(x \mid \omega_i) P(\omega_i) \tag{4.3}$$

The rule in Eq. (4.3) is referred to as Bayes' decision rule. It shows how the observed data *x* changes the decision based on the *prior* probability $P(\omega_i)$ to one based on the posterior probability $P(\omega_i \mid x)$. Decision making based on the *posterior* probability is more reliable, because it employs prior knowledge together with the present observed data. As a matter of fact, when the prior knowledge is non-informative ($P(\omega_1) = P(\omega_2) = P(\omega_3) = 1/3$), the present observed data fully control the decision. On the other hand, when present observed data are ambiguous, then prior knowledge controls the decision. There are many kinds of decision rules based on posterior probability. Our interest is to find the decision rule that leads to minimum overall risk, or minimum error rate in decision.

4.1.1. Minimum-Error-Rate Decision Rules

Bayes' decision rule is designed to minimize the overall risk involved in making a decision. Bayes' decision based on posterior probability $P(\omega_i \mid x)$ instead of prior probability $P(\omega_i)$ is a natural choice. Given an observation *x*, if $P(\omega_k \mid x) \ge P(\omega_i \mid x)$ for all $i \ne k$, we can decide that the true class is ω_j. To justify this procedure, we show such a decision results in minimum decision error.

Let $\Omega = \{\omega_1, ..., \omega_s\}$ be the finite set of *s* possible categories to be predicted and $\Delta = \{\delta_1, ..., \delta_t\}$ be a finite set of *t* possible decisions. Let $l(\delta_i \mid \omega_j)$ be the loss function incurred for making decision δ_i when the true class is ω_j. Using the *prior* probability $P(\omega_i)$ and class-conditional pdf $p(x \mid \omega_i)$, the posterior probability $P(\omega_i \mid x)$ is computed by Bayes' rule as shown in Eq. (4.1). If we make decision δ_i when the true class is ω_j, we incur a loss $l(\delta_i \mid \omega_j)$. Since the posterior probability $P(\omega_j \mid x)$ is the probability that the true class is ω_j after observing the data *x*, the expected loss associated with making decision δ_i is:

$$R(\delta_i \mid x) = \sum_{j=1}^{s} l(\delta_i \mid \omega_j) P(\omega_j \mid x) \tag{4.4}$$

In decision-theoretic terminology, the above expression is called *conditional risks*. The overall risk *R* is the expected loss associated with a given decision rule. The decision rule is employed as a decision function $\delta(x)$ that maps the data *x* to one of the decisions $\Delta = \{\delta_1, ..., \delta_t\}$. Since $R(\delta_i \mid x)$ is the conditional risk associated with decision δ_i, the overall risk is given by:

$$R = \int_{-\infty}^{\infty} R(\delta(x) \mid x) p(x) dx \tag{4.5}$$

If the decision function $\delta(x)$ is chosen so that the conditional risk $R(\delta(x) \mid x)$ is minimized for every *x*, the overall risk is minimized. This leads to the Bayes' decision rule: To minimize the overall risk, we compute the conditional risk shown in Eq. (4.4) for $i = 1, ..., t$ and select the decision δ_i for which the conditional risk $R(\delta_i \mid x)$ is minimum. The resulting minimum overall risk is known as *Bayes' risk* that has the best performance possible.

The loss function $l(\delta_i \mid \omega_j)$ in the Bayes' decision rule can be defined as:

$$l(\delta_i \mid \omega_j) = \begin{cases} 0 & i = j \\ & \qquad i,\ j = 1, \ldots, s \\ 1 & i \neq j \end{cases} \tag{4.6}$$

This loss function assigns no loss to a correct decision where the true class is ω_i and the decision is δ_i, which implies that the true class must be ω_i. It assigns a unit loss to any error where $i \neq j$; i.e., all errors are equally costly. This type of loss function is known as a *symmetrical* or *zero-one* loss function. The risk corresponding to this loss function equals the classification error rate, as shown in the following equation.

$$
\begin{aligned}
R(\delta_i \mid x) &= \sum_{j=1}^{s} l(\delta_i \mid \omega_j) P(\omega_j \mid x) = \sum_{j \neq i} P(\omega_j \mid x) \\
&= \sum_{j=1}^{s} P(\omega_j \mid x) - P(\omega_i \mid x) = 1 - P(\omega_i \mid x)
\end{aligned}
\tag{4.7}
$$

Here $P(\omega_i \mid x)$ is the conditional probability that decision δ_i is correct after observing the data *x*. Therefore, in order to minimize classification error rate, we have to choose the decision of class *i* that maximizes the posterior probability $P(\omega_i \mid x)$. Furthermore, since $p(x)$ is a constant, the decision is equivalent to picking the class *i* that maximizes $p(x \mid \omega_i) P(\omega_i)$. The Bayes' decision rule can be formulated as follows:

$$
\delta(x) = \arg\max_i P(\omega_i \mid x) = \arg\max_i P(x \mid \omega_i) P(\omega_i) \tag{4.8}
$$

This decision rule, which is based on the maximum of the posterior probability $P(\omega_i \mid x)$, is called the *minimum-error-rate decision rule*. It minimizes the classification error rate. Although our description is for random variable *x*, Bayes' decision rule is applicable to multivariate random vector **x** without loss of generality.

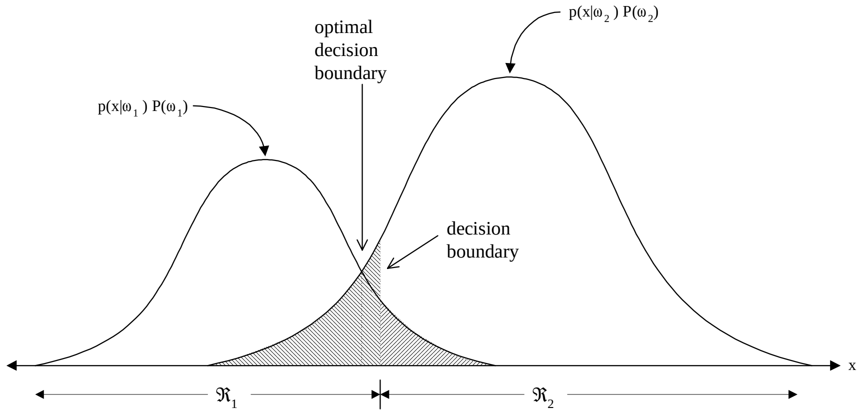

Figure 4.1 Calculation of the likelihood of classification error [22]. The shaded area represents the integral value in Eq. (4.9).

A pattern classifier can be regarded as a device for partitioning the feature space into decision regions. Without loss of generality, we consider a two-class case. Assume that the classifier divides the space $\Re$ into two regions, $\Re_1$ and $\Re_2$. To compute the likelihood of errors, we need to consider two cases. In the first case, *x* falls in $\Re_1$, but the true class is ω_2. In the other case, *x* falls in $\Re_2$, but the true class is ω_1. Since these two cases are mutually exclusive, we have

$$\begin{aligned} P(error) &= P(x \in \Re_1, \omega_2) + P(x \in \Re_2, \omega_1) \\ &= P(x \in \Re_1 \mid \omega_2)P(\omega_2) + P(x \in \Re_2 \mid \omega_1)P(\omega_1) \\ &= \int_{\Re_1} P(x \mid \omega_2)P(\omega_2)dx + \int_{\Re_2} P(x \mid \omega_1)P(\omega_1)dx \end{aligned} \tag{4.9}$$

Figure 4.1 illustrates the calculation of the classification error in Eq. (4.9). The two terms in the summation are merely the tail areas of the function $P(x \mid \omega_i)P(\omega_i)$. It is clear that this decision boundary is not optimal. If we move the decision boundary a little bit to the left, so that the decision is made to choose the class *i* based on the maximum value of $P(x \mid \omega_i)P(\omega_i)$, the tail integral area $P(error)$ becomes minimum, which is the Bayes' decision rule.

4.1.2. Discriminant Functions

The decision problem above can also be viewed as a pattern classification problem where unknown data **x**[1] are classified into known categories, such as the classification of sounds into phonemes using spectral data **x**. A classifier is designed to classify data **x** into *s* categories by using *s* discriminant functions, $d_i(\mathbf{x})$, computing the similarities between the unknown data **x** and each class ω_i and assigning **x** to class ω_j if

$$d_j(\mathbf{x}) > d_i(\mathbf{x}) \quad \forall i \neq j \tag{4.10}$$

This representation of a classifier is illustrated in Figure 4.2.

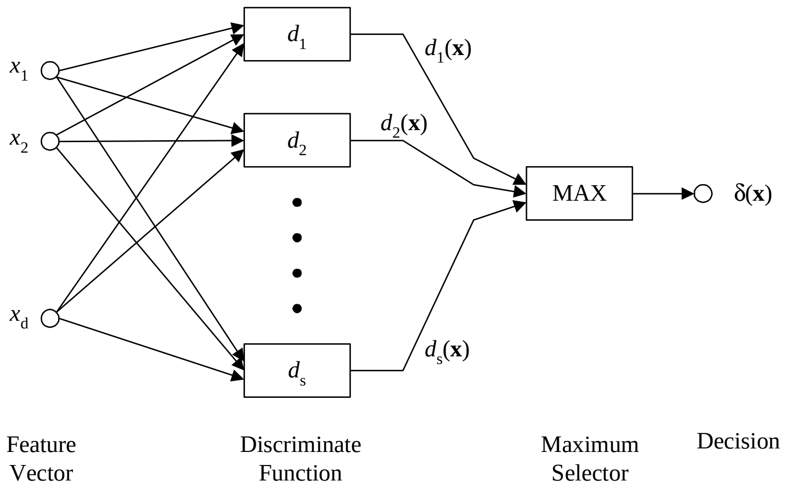

Figure 4.2 Block diagram of a classifier based on discriminant functions [22].

[1] Assuming **x** is a *d*-dimensional vector.

A Bayes' classifier can be represented in the same way. Based on the Bayes' classifier, unknown data **x** are classified on the basis of Bayes' decision rule, which minimizes the conditional risk $R(a_i \mid \mathbf{x})$. Since the classification decision of a pattern classifier is based on the maximum discriminant function shown in Eq. (4.10), we define our discriminant function as:

$$d_i(\mathbf{x}) = -R(a_i \mid \mathbf{x}) \tag{4.11}$$

As such, the maximum discriminant function corresponds to the minimum conditional risk. In the minimum-error-rate classifier, the decision rule is to maximize the posterior probability $P(\omega_i \mid \mathbf{x})$. Thus, the discriminant function can be written as follows:

$$d_i(\mathbf{x}) = P(\omega_i \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \omega_i)P(\omega_i)}{p(\mathbf{x})} = \frac{p(\mathbf{x} \mid \omega_i)P(\omega_i)}{\sum_{j=1}^{s} p(\mathbf{x} \mid \omega_j)P(\omega_j)} \tag{4.12}$$

There is a very interesting relationship between Bayes' decision rule and the hypotheses testing method described in Chapter 3. For a two-class pattern recognition problem, the Bayes' decision rule in Eq. (4.2) can be written as follows:

$$p(\mathbf{x} \mid \omega_1)P(\omega_1) \underset{\omega_2}{\overset{\omega_1}{\gtrless}} p(\mathbf{x} \mid \omega_2)P(\omega_2) \tag{4.13}$$

Eq. (4.13) can be rewritten as:

$$\ell(x) = \frac{p(\mathbf{x} \mid \omega_1)}{p(\mathbf{x} \mid \omega_2)} \underset{\omega_2}{\overset{\omega_1}{\gtrless}} \frac{P(\omega_2)}{P(\omega_1)} \tag{4.14}$$

The term $\ell(x)$ is called *likelihood ratio* and is the basic quantity in hypothesis testing [73]. The term $P(\omega_2)/P(\omega_1)$ is called the *threshold value* of the likelihood ratio for the decision. Often it is convenient to use the log-likelihood ratio instead of the likelihood ratio for the decision rule. Namely, the following single discriminant function can be used instead of $d_1(x)$ and $d_2(x)$ for:

$$d(\mathbf{x}) = \log \ell(\mathbf{x}) = \log p(\mathbf{x} \mid \omega_1) - \log p(\mathbf{x} \mid \omega_2) \underset{\omega_2}{\overset{\omega_1}{\gtrless}} \log P(\omega_2) - \log P(\omega_1) \tag{4.15}$$

As the classifier assigns data **x** to class ω_i, the data space is divided into *s* regions, $\Re_1^d, \Re_2^d, \ldots, \Re_s^d$, called decision regions. The boundaries between decision regions are called decision boundaries and are represented as follows (if they are contiguous):

$$d_i(\mathbf{x}) = d_j(\mathbf{x}) \quad i \neq j \tag{4.16}$$

For points on the decision boundary, the classification can go either way. For a Bayes' classifier, the conditional risk associated with either decision is the same and how to break the tie does not matter. Figure 4.3 illustrates an example of decision boundaries and regions for a three-class classifier on a scalar data sample *x*.

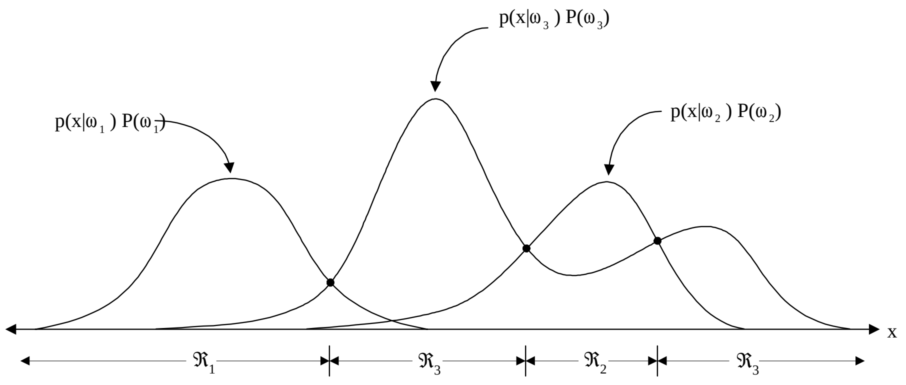
$p(x|\omega_3)\,P(\omega_3)$
$p(x|\omega_1)\,P(\omega_1)$
$p(x|\omega_2)\,P(\omega_2)$
x
$\Re_1$
$\Re_3$
$\Re_2$
$\Re_3$

Figure 4.3 An example of decision boundaries and regions. For simplicity, we use scalar variable *x* instead of a multi-dimensional vector [22].

4.2. How to Construct Classifiers

In the Bayes classifier, or the minimum-error-rate classifier, the prior probability $P(\omega_i)$ and class-conditional pdf $p(\mathbf{x}\,|\,\omega_i)$ are known. Unfortunately, in pattern recognition domains, we rarely have complete knowledge of class-conditional pdf's and/or prior probability $P(\omega_i)$. They often must be estimated or learned from the training data. In practice, the estimation of the prior probabilities is relatively easy. Estimation of the class-conditional pdf is more complicated. There is always concern to have sufficient training data relative to the tractability of the huge dimensionality of the sample data **x**. In this chapter we focus on estimation methods for the class-conditional pdf.

The estimation of the class-conditional pdfs can be nonparametric or parametric. In nonparametric estimation, no model structure is assumed and the pdf is directly estimated from the training data. When large amounts of sample data are available, nonparametric learning can accurately reflect the underlying probabilistic structure of the training data. However, available sample data are normally limited in practice, and parametric learning can achieve better estimates if valid model assumptions are made. In parametric learning, some general knowledge about the problem space allows one to parameterize the class-conditional pdf, so the severity of sparse training data can be reduced significantly. Suppose the pdf $p(\mathbf{x}\,|\,\omega_i)$ is assumed to have a certain probabilistic structure, such as the Gaussian pdf. In such cases, only the mean vector $\boldsymbol{\mu}_i$ (or mean μ_i) and covariance matrix Σ_i (or variance σ^2) need to be estimated.

When the observed data *x* only takes discrete values from a finite set of *N* values, the class-conditional pdf is often assumed nonparametric, so there will be $N-1$ free parameters in the probability function $p(\mathbf{x}|\omega_i)$[2]. When the observed data **x** takes continuous values, parametric approaches are usually necessary. In many systems, the continuous class-conditional pdf (likelihood) $p(\mathbf{x}|\omega_i)$ is assumed to be a Gaussian distribution or a mixture of Gaussian distributions.

In pattern recognition, the set of data samples, which is often collected to estimate the parameters of the recognizer (including the prior and class-conditional pdf), is referred to as the *training set*. In contrast to the training set, the *testing set* is referred to the independent set of data samples, which is used to evaluate the recognition performance of the recognizer.

For parameter estimation or learning, it is also important to distinguish between *supervised learning* and *unsupervised learning*. Let's denote the pair $(\mathbf{x},\omega)$ as a sample, where **x** is the observed data and ω is the class from which the data **x** comes. From the definition, it is clear that $(\mathbf{x},\omega)$ are jointly distributed random variables. In supervised learning, ω, information about the class of the sample data **x** is given. Such sample data are usually called labeled data or complete data, in contrast to incomplete data where the class information ω is missing for unsupervised learning. Techniques for parametric unsupervised learning are discussed in Section 4.4.

In Chapter 3 we introduced two most popular parameter estimation techniques – *maximum likelihood estimation* (MLE) and *maximum a posteriori probability estimation* (MAP). Both MLE and MAP are supervised learning methods since the class information is required. MLE is the most widely used because of its efficiency. The goal of MLE is to find the set of parameters that maximizes the probability of generating the training data actually observed. The class-conditional pdf is typically parameterized. Let $\mathbf{\Phi}_i$ denote the parameter vector for class *i*. We can represent the class-conditional pdf as a function of $\mathbf{\Phi}_i$ as $p(\mathbf{x}|\omega_i,\mathbf{\Phi}_i)$. As stated earlier, in supervised learning, the class name ω_i is given for each sample data in training set $\{\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_n\}$. We need to make an assumption[3] that samples in class ω_i give no information about the parameter vector $\mathbf{\Phi}_j$ of the other class ω_j. This assumption allows us to deal with each class independently, since the parameter vectors for different categories are functionally independent. The class-conditional pdf can be rewritten as $p(\mathbf{x}|\mathbf{\Phi})$, where $\mathbf{\Phi}=\{\mathbf{\Phi}_1,\mathbf{\Phi}_i,\ldots,\mathbf{\Phi}_n\}$. If a set of random samples $\{\mathbf{X}_1,\mathbf{X}_2,\ldots,\mathbf{X}_n\}$ is drawn independently according to a pdf $p(\mathbf{x}|\mathbf{\Phi})$, where the value of the parameter $\mathbf{\Phi}$ is unknown, the MLE method described in Chapter 3 can be directly applied to estimate $\mathbf{\Phi}$.

Similarly, MAP estimation can be applied to estimate $\mathbf{\Phi}$ if knowledge about a prior distribution is available. In general, MLE is used for estimating parameters from scratch without any prior knowledge, and MAP estimation is used for parameter adaptation where

[2] Since all the probabilities need to add up to one.

[3] This assumption is only true for non-discriminant estimation. Samples in class ω_i may affect parameter vector $\mathbf{\Phi}_i$ of the other classes in discriminant estimation methods as described in Section 4.3

the behavior of prior distribution is known and only small amount adaptation data is available. When the amount of adaptation data increases, MAP estimation converges to MLE.

4.2.1. Gaussian Classifiers

A *Gaussian classifier* is a Bayes' classifier where class-conditional probability density $p(\mathbf{x} | \omega_i)$ for each class ω_i is assumed to have a Gaussian distribution[4]:

$$p(\mathbf{x} | \omega_i) = \frac{1}{(2\pi)^{n/2} |\Sigma_i|^{1/2}} \exp\left[-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_i)^t \Sigma_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) \right] \tag{4.17}$$

As discussed in Chapter 3, the parameter estimation techniques are well suited for the Gaussian family. The MLE of the Gaussian parameter is just its sample mean and variance (or co-variance matrix). A Gaussian classifier is equivalent to the one using a quadratic discriminant function. As noted in Eq. (4.12), the discriminant function for a Bayes' decision rule is the posterior probability $p(\omega_i | \mathbf{x})$ or $p(\mathbf{x} | \omega_i) P(\omega_i)$ as the discriminant function. Assuming $p(\mathbf{x} | \omega_i)$ is a multivariate Gaussian density as shown in Eq. (4.17), a discriminant function can be written as follows:

$$\begin{aligned} d_i(\mathbf{x}) &= \log p(\mathbf{x} | \omega_i) p(\omega_i) \\ &= -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_i)^t \Sigma_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) + \log p(\omega_i) - \frac{1}{2} \log |\Sigma| - \frac{d}{2} \log 2\pi \end{aligned} \tag{4.18}$$

If we have a uniform prior $p(\omega_i)$, it is clear that the above discriminant function $d_i(\mathbf{x})$ is a quadratic function. Once we have the *s* Gaussian discriminant functions, the decision process simply assigns data **x** to class ω_j iff

$$j = \arg\max_i d_i(\mathbf{x}) \tag{4.19}$$

When all the Gaussian pdfs have the same covariance matrix ($\Sigma_i = \Sigma$ for $i = 1, 2, \ldots, s$), the quadratic term $\mathbf{x}^t \Sigma^{-1} \mathbf{x}$ is independent of the class and can be treated as a constant. Thus the following new discriminant function $d_i(\mathbf{x})$ can be used [22];

$$d_i(\mathbf{x}) = \mathbf{a}_i^t \mathbf{x} + \mathbf{c}_i \tag{4.20}$$

where $\mathbf{a}_i = \Sigma^{-1} \boldsymbol{\mu}_i$ and $\mathbf{c}_i = -\frac{1}{2} \boldsymbol{\mu}_i^t \Sigma_i^{-1} \boldsymbol{\mu}_i + \log P(\omega_i)$. $d_i(\mathbf{x})$ in Eq. (4.20) is a linear discriminant function. For linear discriminant functions, the decision boundaries are hyperplanes. For the two-class case (ω_1 and ω_2), and assuming that data sample **x** is a real random vector, the decision boundary can be shown to be the following hyperplane:

[4] The Gaussian distribution may include a mixture of Gaussian pdfs.

$$\mathbf{A}^t(\mathbf{x}-\mathbf{b})=0 \tag{4.21}$$

where

$$\mathbf{A}=\Sigma^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2) \tag{4.22}$$

and

$$\mathbf{b}=\frac{1}{2}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2)-\frac{(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2)\log\left[P(\omega_1/\omega_2\right]}{(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2)^t\Sigma^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2)} \tag{4.23}$$

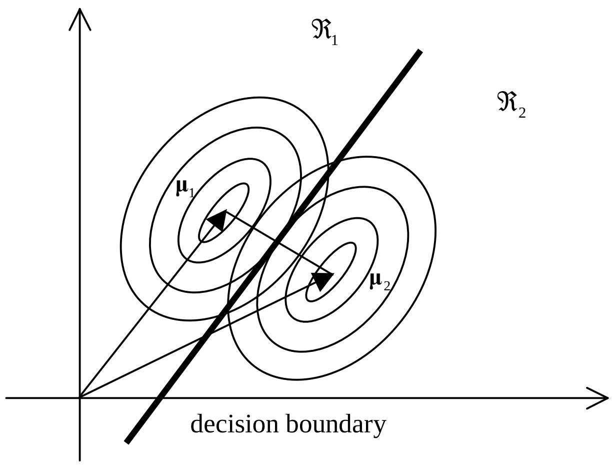

Figure 4.4 Decision boundary for a two-class Gaussian classifier. Gaussian distributions for the two categories have the same covariance matrix Σ. Each ellipse represents the region with the same likelihood probability [22].

Figure 4.4 shows a two dimensional decision boundary for a two-class Gaussian classifier with the same covariance matrix. Please note that the decision hyperplane is generally not orthogonal to the line between the means $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$, although it does intersect that line at the point **b**, which is halfway between $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$. The analysis above is based on the case of uniform priors ($p(\omega_1)=p(\omega_2)$). For nonuniform priors, the decision hyperplane moves away from the more likely mean.

Finally, if each dimension of random vector **x** is statistically independent and has the same variance σ^2, i.e., $\Sigma_1=\Sigma_2=\sigma^2\mathbf{I}$, Figure 4.4 becomes Figure 4.5. The ellipse in Figure 4.4 becomes a circle because the variance σ^2 is the same for all dimensions [22].

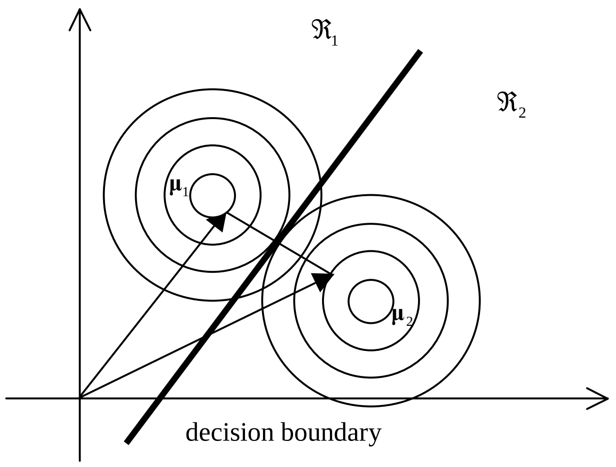

Figure 4.5 Decision boundary for a two-class Gaussian classifier. Gaussian distributions for the two categories have the same covariance matrix $\sigma^2\mathbf{I}$. Each circle represents the region with the same likelihood probability [22].

4.2.2. The Curse of Dimensionality

More features (higher dimensions for sample **x**) and more parameters for the class-conditional pdf $p(\mathbf{x}\,|\,\Phi)$ may lead to lower classification error rate. If the features are statistically independent, there are theoretical arguments that support better classification performance with more features. Let us consider a simple two-class Gaussian classifier. Suppose the prior probabilities $p(\omega_i)$ are equal and the class-conditional Gaussian pdf's $p(\mathbf{x}\,|\,\mathbf{m}_i,\Sigma_i)$ share the same covariance matrix Σ. According to Eqs. (4.9) and (4.21), the Bayes' classification error rate is given by:

$$\begin{aligned} P(error) &= 2\int_{\Re_2} P(\mathbf{x}\,|\,\omega_1)P(\omega_1)d\mathbf{x} \\ &= 2\int_{\mathbf{A}^t(\mathbf{x}-\mathbf{b})=0}^{\infty} \frac{1}{(2\pi)^{n/2}|\Sigma_i|^{1/2}} \exp\left[-\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^t\Sigma_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)\right]d\mathbf{x} \\ &= \frac{1}{\sqrt{2\pi}}\int_{\frac{r}{2}}^{\infty} e^{\frac{-1}{2}z^2}dz \end{aligned} \tag{4.24}$$

where $r=\sqrt{(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2)^t\Sigma^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2)}$. When features are independent, the covariance matrix becomes a diagonal one. The following equation shows that each independent feature helps to reduce the error rate[5]:

[5] When the means of a feature for the two classes are exactly the same, adding such a feature does not reduce the Bayes' error. Nonetheless, according to Eq. (4.25), the Bayes' error cannot possibly be increased by incorporating an additional independent feature.

$$r = \sqrt{\sum_{i=1}^{d} \left(\frac{\mu_{1i} - \mu_{2i}}{\sigma_i} \right)^2} \tag{4.25}$$

where μ_{1i} and μ_{2i} are the i^{th} -dimension of mean vectors $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_1$ respectively.

Unfortunately, in practice, the inclusion of additional features may lead to worse classification results. This paradox is called *the curse of dimensionality*. The fundamental issue, called *trainability*, refers to how well the parameters of the classifier are trained from the limited training samples. Trainability can be illustrated by a typical curve-fitting (or regression) problem. Figure 4.6 shows a set of eleven data points and several polynomial fitting curves with different orders. Both the first-order (linear) and second-order (quadratic) polynomials shown provide fairly good fittings for these data points. Although the tenth-order polynomial fits the data points perfectly, no one would expect such an under-determined solution to fit the new data well. In general, many more data samples would be necessary to get a good estimate of a tenth-order polynomial than of a second-order polynomial, because reliable interpolation or extrapolation can be attained only with an over-determined solution.

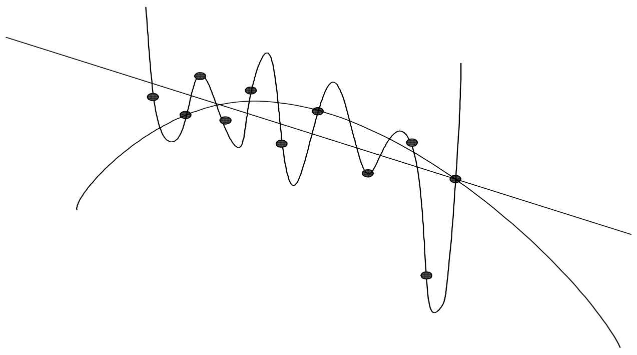

Figure 4.6 Fitting eleven data points with polynomial functions of different orders [22].

Figure 4.7 shows the error rates for two-phonemes (/*ae*/ and /*ih*/) classification where two phonemes are modeled by mixtures of Gaussian distributions. The parameters of mixtures of Gaussian are trained from a varied set of training samples via maximum likelihood estimation. The curve illustrates the classification error rate as a function of the number of training samples and the number of mixtures. For every curve associated with a finite number of samples, there are an optimal number of mixtures. This illustrates the importance of trainability: it is critical to assure there are enough samples for training an increased number of features or parameters. When the size of training data is fixed, increasing the number of features or parameters beyond a certain point is likely to be counterproductive.

When you have an insufficient amount of data to estimate the parameters, some simplification can be made to the structure of your models. In general, the estimation for higher-order statistics, like variances or co-variance matrices, requires more data than that for lower- order statistics, like mean vectors. Thus more attention often is paid to dealing with the estimation of covariance matrices. Some frequent used heuristics for Gaussian distributions include the use of the same covariance matrix for all mixture components [77], diagonal covariance matrix, and shrinkage (also referred to as regularized discriminant analysis), where the covariance matrix is interpolated with the constant covariance matrix [23, 50].

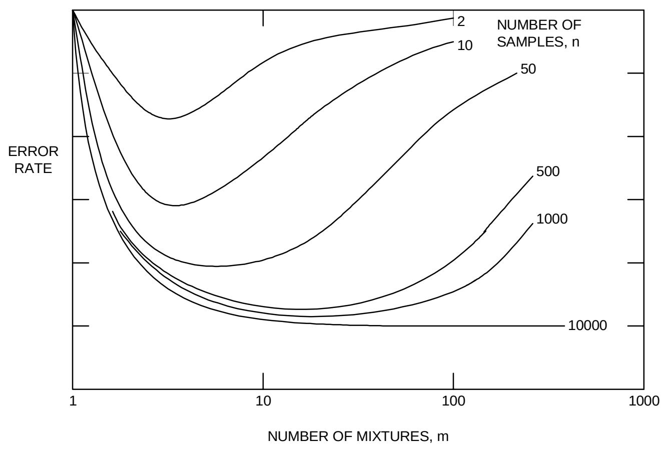

Figure 4.7 Two-phoneme (/*ae*/ and /*ih*/) classification results as a function of the number of Gaussian mixtures and the number of training samples.

4.2.3. Estimating the Error Rate

Estimating the error rate of a classifier is important. We want to see whether the classifier is good enough to be useful for our task. For example, telephone applications show that some accuracy is required before users would switch from using the touch-tone to the speech recognizer. It is also critical to compare the performance of a classifier (algorithm) against an alternative. In this section we deal with how to estimate the true classification error rate.

One approach is to compute the theoretic error rate from the parametric model as shown in Eq. (4.25). However, there are several problems with this approach. First, such an approach almost always under-estimates, because the parameters estimated from the training samples might not be realistic unless the training samples are representative and sufficient. Second, all the assumptions about models and distributions might be severely wrong. Finally, it is very difficult to compute the exact error rate, as in the simple case illustrated in Eq. (4.25).

Instead, you can estimate the error rate empirically. In general, the recognition error rate on the training set should be viewed only as a lower bound, because the estimate can be made to minimize the error rate on the training data. Therefore, a better estimate of the recognition performance should be obtained on an independent test set. The question now is how representative is the error rate computed from an arbitrary independent test set. The common process of using some of the data samples for design and reserving the rest for test is called the *holdout* or *H* method.

Suppose the true but unknown classification error rate of the classifier is *p*, and one observes that *k* out of *n* independent randomly drawn test samples are misclassified. The random variable *K* should have a binomial distribution $B(n, p)$. The maximum likelihood estimation for *p* should be

$$\hat{p} = \frac{k}{n} \tag{4.26}$$

The statistical test for binomial distribution is discussed in Chapter 3. For a 0.05 significance level, we can compute the following equations to get the range (p_1, p_2):

$$2P(k \le m \le n) = 2\sum_{m=k}^{n} \binom{n}{m} (p_1)^m (1-p_1)^{n-m} = 0.05 \quad \text{when } k > np_1 \tag{4.27}$$

$$2P(0 \le m \le k) = 2\sum_{m=0}^{k} \binom{n}{m} (p_2)^m (1-p_2)^{n-m} = 0.05 \quad \text{when } k < np_2 \tag{4.28}$$

Equations (4.27) and (4.28) are cumbersome to solve, so the normal test described in Chapter 3 can be used instead. The null hypothesis H_0 is

$H_0 \;:\; p = \hat{p}$

We can use the normal test to find the two boundary points p_1 and p_2 at which we would not reject the null hypothesis H_0.

The range (p_1, p_2) is called the 0.95 confidence intervals because one can be 95% confident that the true error rate *p* falls in the range (p_1, p_2). Figure 4.8 illustrates 95% confidence intervals as a function of $\hat{p}$ and *n*. The curve certainly agrees with our intuition – the larger the number of test samples *n*, the more confidence we have in the MLE estimated error rate $\hat{p}$; otherwise, the $\hat{p}$ can be used only with caution.

Based on the description in the previous paragraph, the larger the test set is, the better it represents the recognition performance of possible data. On one hand, we need more training data to build a reliable and consistent estimate. On the other hand, we need a large independent test set to derive a good estimate of the true recognition performance. This creates a contradictory situation for dividing the available data set into training and independent test set. One way to effectively use the available database is *V-fold cross validation*. It first splits the entire database into *V* equal parts. Each part is used in turn as an independent test set while the remaining (*V* - 1) parts are used for training. The error rate can then be better estimated by averaging the error rates evaluated on the *V* different testing sets. Thus, each part can contribute to both training and test sets during *V*-fold cross validation. This procedure, also called the *leave-one-out* or *U* method [53], is particularly attractive when the number of available samples are limited.

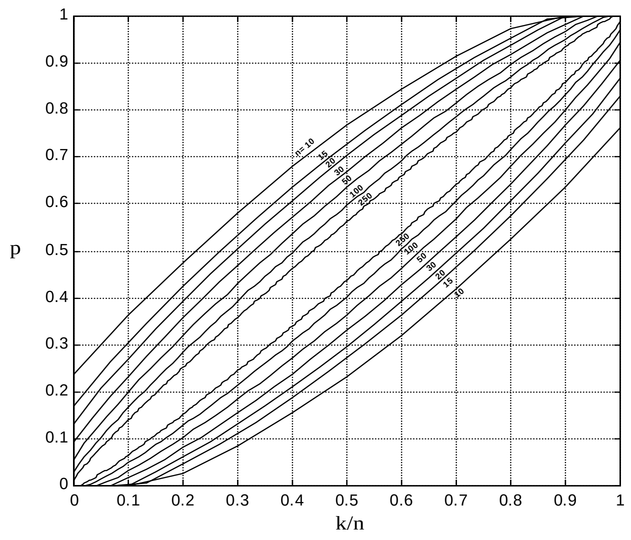

Figure 4.8 95% confidence intervals for classification error rate estimation with normal test

4.2.4. Comparing Classifiers

Given so many design alternatives, it is critical to compare the performance of different classifiers so that the best classifier can be used for real-world applications. It is common for designers to test two classifiers on some test samples and decide if one is superior to the other. Relative efficacy can be claimed only if the difference in performance is statistically significant. In other words, we establish the null hypothesis H_0 that the two classifiers have the same error rates. Based on the observed error patterns, we decide whether we could reject H_0 at the 0.05 level of significance. The test for different classifiers falls into the category of *matched-pairs* tests described in Chapter 3. Classifiers are compared with the same test samples.

We present an effective matched-pairs test - McNemar's test [66] which is particularly suitable for comparing classification results. Suppose there are two classifiers: Q_1 and Q_2. The estimated classification error rates on the same test set for these two classifiers are p_1 and p_2 respectively. The null hypothesis H_0 is $p_1 = p_2$. The classification performance of the two classifiers can be summarized as in Table 4.1. We define q_{ij}'s as follows:

$$q_{00} = P(Q_1 \text{ and } Q_2 \text{ classify data sample correctly})$$

$$q_{01} = P(Q_1 \text{ classifies data sample correctly, but } Q_2 \text{ incorrectly})$$

$$q_{10} = P(Q_2 \text{ classifies data sample correctly, but } Q_1 \text{ incorrectly})$$
$$q_{11} = P(Q_1 \text{ and } Q_2 \text{ classify data sample incorrectly})$$

Table 4.1 Classification performance table for classifier Q_1 and Q_2. N_{00} is the number of samples which Q_1 and Q_2 classify correctly, N_{01} is the number of samples which Q_1 classifies correctly, but Q_2 incorrectly, N_{10} is the of samples which Q_2 classifies correctly, but Q_1 incorrectly, and N_{11} is the number of samples which Q_1 and Q_2 classify incorrectly [30].

		Q_2	
		Correct	Incorrect
Q_1	Correct	N_{00}	N_{01}
	Incorrect	N_{10}	N_{11}

The null hypothesis H_0 is equivalent to H_0^1 : $q_{01} = q_{10}$. If we define $q = q_{10}/(q_{01}+q_{10})$, H_0 is equivalent to H_0^2 : $q = 1/2$. H_0^2 represents the hypothesis that, given only one of the classifiers makes an error, it is equally likely to be either one. We can test H_0^2 based on the data samples on which only one of the classifiers made an error. Let $n = N_{01} + N_{10}$. The observed random variable N_{01} should have a binomial distribution $B(n, 1/2)$. Therefore, the normal test (z-test) described in Chapter 3 can be applied directly to test the null hypothesis H_0^2.

The above procedure is called the *McNemar's test* [66]. If we view the classification results as *N* (the total number of test samples) independent matched pairs, the sign test as described in Chapter 3 can be directly applied to test the null hypothesis that classifier Q_1 is not better than classifier Q_2, that is, the probability that classifier Q_1 performs better than classifier Q_2, *p*, is smaller than or equal to ½.

McNemar's test is applicable when the errors made by a classifier are independent among different test samples. Although this condition is true for most static pattern recognition problems, it is not the case for most speech recognition problems. In speech recognition, the errors are highly inter-dependent because of the use of higher-order language models (described in Chapter 11).

The solution is to divide the test data stream into segments in such a way that errors in one segment are statistically independent of errors in any other segment [30]. A natural candidate for such a segment is a sentence or a phrase after which the speaker pauses. Let N_1^i the number of errors[6] made on the ith segment by classifier Q_1 and N_2^i be the number of errors made on the *i*th segment by classifier Q_2. Under this formulation, the magnitude-difference test described in Chapter 3 can be applied directly to test the null hypothesis that

[6] The errors for speech recognition include substitutions, insertions and deletions as discussed in Chapter 8.

classifiers Q_1 and Q_2 have on the average the same error rate on the pairs of *n* independent segments.

4.3. DISCRIMINATIVE TRAINING

Both MLE and MAP criteria maximize the probability of the model associated with the corresponding data. Only data labeled as belonging to class ω_i are used to train the parameters. There is no guarantee that the observed data **x** from class ω_i actually have a higher likelihood probability $P(\mathbf{x} \mid \omega_i)$ than the likelihood $P(\mathbf{x} \mid \omega_j)$ associated with class *j*, given $j \neq i$. Models generated by MLE or MAP have a loose discriminant nature. Several estimation methods aim for maximum discrimination among models to achieve best pattern recognition performance.

4.3.1. Maximum Mutual Information Estimation

The pattern recognition problem can be formalized as an information channel, as illustrated in Figure 4.9. The source symbol ω is encoded into data **x** and transmitted through an information channel to the observer. The observer utilizes pattern recognition techniques to decode **x** into source symbol $\hat{\omega}$**.** Consistent with the goal of communication channels, the observer hopes the decoded symbol $\hat{\omega}$ is the same as the original source symbol ω**.** Maximum mutual information estimation tries to improve channel quality between input and output symbols.

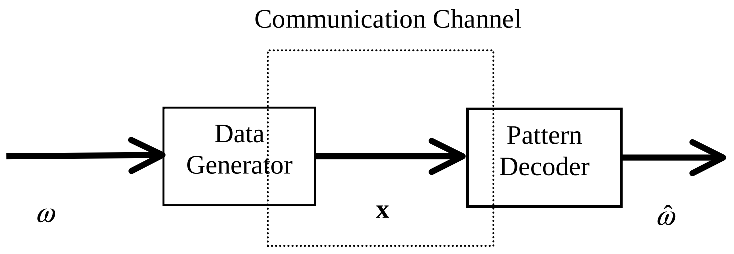

Figure 4.9 An information channel framework for pattern recognition.

As described in Section 4.1.1, the decision rule for the minimum-error-rate classifier selects the class ω_i with maximum posterior probability $P(\omega_i \mid \mathbf{x})$. It is a good criterion to maximize the posterior probability $P(\omega_i \mid \mathbf{x})$ for parameter estimation. Recalling Bayes' rule in Section 4.1, the posterior probability $p(\omega_i \mid \mathbf{x})$ (assuming *x* belongs to class ω_i) is:

$$P(\omega_i \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \omega_i)P(\omega_i)}{p(\mathbf{x})} \tag{4.29}$$

and $p(\mathbf{x})$ can be expressed as follows:

$$p(\mathbf{x}) = \sum_k p(\mathbf{x} \mid \omega_k) p(\omega_k) \tag{4.30}$$

In the classification stage, $p(\mathbf{x})$ can be considered as a constant. However, during training, the value of $p(\mathbf{x})$ depends on the parameters of all models and is different for different $\mathbf{x}$. Equation (4.29) is referred to as *conditional likelihood*. A conditional maximum likelihood estimator (CMLE) θ_{CMLE} is defined as follows:

$$\theta_{CMLE}(\mathbf{x}) = \Phi_{MAP} = \underset{\Phi}{\operatorname{argmax}}\ p_{\Phi}(\omega_i \mid \mathbf{x}) \tag{4.31}$$

The summation in Eq. (4.30) extends over all possible classes that include the correct model and all the possible competing models. The parameter vector $\mathbf{\Phi}$ in Eq. (4.31) includes not only the parameter $\mathbf{\Phi}_i$ corresponding to class ω_i, but also those for all other classes.

Note that in Chapter 3, the mutual information between random variable $\mathbf{X}$ (observed data) and $\mathbf{\Omega}$ (class label) is defined as:

$$I(\mathbf{X},\Omega) = E\left(\log \frac{p(\mathbf{X},\Omega)}{p(\mathbf{X})P(\Omega)}\right) = E\left(\log \frac{p(\mathbf{X} \mid \Omega)P(\Omega)}{p(\mathbf{X})P(\Omega)}\right) \tag{4.32}$$

Since we don't know the probability distribution for $p(\mathbf{X},\Omega)$, we assume our sample $(\mathbf{x},\omega_i)$ is representative and define the following *instantaneous mutual information*:

$$I(\mathbf{x},\omega_i) = \log \frac{p(\mathbf{x},\omega_i)}{p(\mathbf{x})P(\omega_i)} \tag{4.33}$$

If equal prior $p(\omega_i)$ is assumed for all classes, maximizing the conditional likelihood in Eq. (4.29) is equivalent to maximizing the mutual information defined in Eq. (4.33). CMLE becomes *maximum mutual information estimation* (MMIE). It is important to note that, in contrast to MLE, MMIE is concerned with distributions over all possible classes. Equation (4.30) can be rewritten as two terms, one corresponding to the correct one, and the other corresponding to the competing models:

$$p(\mathbf{x}) = p(\mathbf{x} \mid \omega_i)P(\omega_i) + \sum_{k \neq i} p(\mathbf{x} \mid \omega_k)P(\omega_k) \tag{4.34}$$

Based on the new expression of $p(\mathbf{x})$ shown in Eq. (4.34), the posterior probability $p(\omega_i \mid \mathbf{x})$ in Eq. (4.29) can be rewritten as:

$$P(\omega_i \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \omega_i)P(\omega_i)}{p(\mathbf{x} \mid \omega_i)P(\omega_i) + \sum_{k \neq i} p(\mathbf{x} \mid \omega_k)P(\omega_k)} \tag{4.35}$$

Now, maximization of the posterior probability $p(\omega_i \mid \mathbf{x})$ with respect to all models leads to a discriminant model.[7] It implies that the contribution of $p(\mathbf{x} \mid \omega_i)P(\omega_i)$ from the true model needs to be enforced, while the contribution of all the competing models, specified by $\sum_{k \neq i} p(\mathbf{x} \mid \omega_k)P(\omega_k)$, needs to be minimized. Maximization of Eq. (4.35) can be further rewritten as:

$$P(\omega_i \mid \mathbf{x}) = \frac{1}{1 + \frac{\sum_{k \neq i} p(\mathbf{x} \mid \omega_i) p(\omega_i)}{p(\mathbf{x} \mid \omega_i) p(\omega_i)}} \tag{4.36}$$

Maximization is thus equivalent to maximization of the following term, which is clearly a discriminant criterion between model ω_i and the sum of all other competing models.

$$\frac{p(\mathbf{x} \mid \omega_i) p(\omega_i)}{\sum_{k \neq i} p(\mathbf{x} \mid \omega_k) p(\omega_k)} \tag{4.37}$$

Equation (4.37) also illustrates a fundamental difference between MLE and MMIE. In MLE, only the correct model needs to be updated during training. However, every MMIE model is updated even with one training sample. Furthermore, the greater the prior probability $p(\omega_k)$ for class ω_k, the more effect it has on the maximum mutual information estimator θ_{MMIE}. This makes sense, since the greater the prior probability $p(\omega_k)$, the greater the chance for the recognition system to mis-recognize ω_i as ω_k. MLE is a simplified version of MMIE by restricting the training of model using the data for the model only. This simplification allows the denominator term in Eq. (4.35) to contain the correct model so that it can be dropped as a constant term. Thus, maximization of the posterior probability $p(\omega_i \mid \mathbf{x})$ can be transformed into maximization of the likelihood $p(\mathbf{x} \mid \omega_i)$.

Although likelihood and posterior probability are transformable based on Bayes' rule, MLE and MMIE often generate different results. Discriminative criteria like MMIE attempt to achieve minimum error rate. It might actually produce lower likelihood than for the underlying probability density $p(\mathbf{x} \mid \omega_k)$. However, if the assumption of the underlying distributions is correct and there are enough (or infinite) training data, the estimates should converge to the true underlying distributions. Therefore, Bayes' rule should be satisfied and MLE and MMIE should produce the same estimate.

Arthur Nadas [71] showed that if the prior distribution (language model) and the assumed likelihood distribution family are correct, both MLE and MMIE are consistent estimators, but MMIE has a greater variance. However, when some of those premises are not valid, it is desirable to use MMIE to find the estimate that maximizes the mutual information (instead of likelihood) between sample data and its class information. The difference be-

[7] General minimum-error-rate estimation is described in Section 4.3.2.

tween these two estimation techniques is that MMIE not only aims to increase the likelihood for the correct class, but also tries to decrease the likelihood for the incorrect classes. Thus, MMIE in general possesses more discriminating power among different categories. Although MMIE is theoretically appealing, computationally it is very expensive. Comparing with MLE, every data sample needs to train all the possible models instead of the corresponding model. It also lacks an efficient maximization algorithm. You need to use a gradient descent algorithm.

4.3.1.1. Gradient Descent

To maximize Eq. (4.37) over the entire parameter space $\mathbf{\Phi} = \{\mathbf{\Phi}_1, \mathbf{\Phi}_2, \ldots, \mathbf{\Phi}_M\}$ with M classes, we define the mutual information term in Eq. (4.37) to be a function of Φ. To fit into the traditional gradient descent framework, we take the inverse of Eq. (4.37) as our optimization function to minimize the following function:[8]

$$F(\mathbf{\Phi}) = \frac{1}{P_{\mathbf{\Phi}}(\omega_i \mid \mathbf{x})} = \frac{\sum_{k \neq i} p_{\mathbf{\Phi}_k}(\mathbf{x} \mid \omega_k) p(\omega_k)}{p_{\mathbf{\Phi}_i}(\mathbf{x} \mid \omega_i) p(\omega_i)} \tag{4.38}$$

The gradient descent algorithm starts with some initial estimate $\mathbf{\Phi}^0$ and computes the gradient vector $\nabla F(\mathbf{\Phi})$ (∇ is defined in Chapter 3). It obtains a new estimate $\mathbf{\Phi}^1$ by moving $\mathbf{\Phi}^0$ in the direction of the steepest descent, i.e., along the negative of the gradient. Once it obtains the new estimate, it can perform the same gradient descent procedure iteratively until $F(\mathbf{\Phi})$ converges to the local minimum. In summary, it obtains $\mathbf{\Phi}^{t+1}$ from $\mathbf{\Phi}^t$ by the following formula:

$$\Phi^{t+1} = \Phi^t - \varepsilon_t \nabla F(\Phi) |_{\Phi = \Phi^t} \tag{4.39}$$

where ε_t is the learning rate (or step size) for the gradient descent.

Why can gradient descent lead $F(\mathbf{\Phi})$ to a local minimum? Based on the definition of gradient vector, $F(\mathbf{\Phi})$ can be approximated by the first order expansion if the correction term $\Delta\mathbf{\Phi}$ is small enough.

$$F(\mathbf{\Phi}^{t+1}) \approx F(\mathbf{\Phi}^t) + \Delta\mathbf{\Phi} * \nabla F(\mathbf{\Phi}) |_{\mathbf{\Phi} = \mathbf{\Phi}^t} \tag{4.40}$$

$\Delta\Phi$ can be expressed as the following term based on Eq. (4.39)

$$\Delta\mathbf{\Phi} = \mathbf{\Phi}^{t+1} - \mathbf{\Phi}^t = -\varepsilon_t \nabla F(\mathbf{\Phi}) |_{\mathbf{\Phi} = \mathbf{\Phi}^t} \tag{4.41}$$

Thus, we can obtain the following equation:

[8] You can use the logarithm of the object function to make it easier to compute the derivative in gradient descent.

$$F(\mathbf{\Phi}^{t+1}) - F(\mathbf{\Phi}^t) = -\varepsilon_t \left\langle \nabla F(\mathbf{\Phi})\,|_{\mathbf{\Phi}=\mathbf{\Phi}^t}, \nabla F(\Phi)\,|_{\mathbf{\Phi}=\mathbf{\Phi}^t} \right\rangle$$
$$= -\varepsilon_t \left\| \nabla F(\mathbf{\Phi})\,|_{\mathbf{\Phi}=\mathbf{\Phi}^t} \right\|^2 < 0 \quad (4.42)$$

where $\langle x, y \rangle$ represents the inner product of two vectors, and $\|x\|$ represents the Euclidean norm of the vector. Equation (4.42) means that the gradient descent finds a new estimate $\mathbf{\Phi}^{t+1}$ that makes the value of the function $F(\mathbf{\Phi})$ decrease.

The gradient descent algorithm needs to go through an iterative hill-climbing procedure to converge to the local minimum (estimate). Gradient descent usually requires many iterations to converge. The algorithm usually stops when the change of the parameter $\Delta\Phi$ becomes small enough. That is,

$$\varepsilon_t \nabla F(\mathbf{\Phi})\,|_{\mathbf{\Phi}=\mathbf{\Phi}^t} < \lambda \quad (4.43)$$

where λ is a preset threshold.

Based on the derivation above, the learning rate coefficient ε_t must be small enough for gradient descent to converge. However, if ε_t is too small, convergence is needlessly slow. Thus, it is very important to choose an appropriate ε_t. It is proved [47] [48] that gradient converges almost surely if the learning rate coefficient ε_t satisfies the following condition:

$$\sum_{t=0}^{\infty} \varepsilon_t = \infty, \quad \sum_{t=0}^{\infty} \varepsilon_t^{\,2} < \infty, \text{ and } \varepsilon_t > 0 \quad (4.44)$$

One popular choice of ε_t satisfying the above condition is

$$\varepsilon_t = \frac{1}{t+1} \quad (4.45)$$

Another way to find an appropriate ε_t is through the second-order expansion:

$$F(\mathbf{\Phi}^{t+1}) \approx F(\mathbf{\Phi}^t) + \Delta\mathbf{\Phi} \nabla F(\mathbf{\Phi})\,|_{\mathbf{\Phi}=\mathbf{\Phi}^t} + \frac{1}{2}(\Delta\mathbf{\Phi})^t \mathbf{D} \Delta\mathbf{\Phi} \quad (4.46)$$

where **D** is the *Hessian* matrix [23] of the second-order gradient operator where the *i*-th row and *j*-th element $D_{i,j}$ are given by the following partial derivative:

$$D_{i,j} = \frac{\partial^2 F(\mathbf{\Phi})}{\partial \mathbf{\Phi}_i \partial \mathbf{\Phi}_j} \quad (4.47)$$

By substituting $\Delta\Phi$ from Eq. (4.41) into Eq. (4.46), we can obtain

$$F(\mathbf{\Phi}^{t+1}) \approx F(\mathbf{\Phi}^t) - \varepsilon_t \|\nabla F\|^2 + \frac{1}{2}\varepsilon_t^2 \nabla F^t \mathbf{D} \nabla F \quad (4.48)$$

From this, it follows that ε_t can be chosen as follows to minimize $F(\mathbf{\Phi})$ [23]:

$$\varepsilon_t = \frac{\|\nabla F\|^2}{\nabla F^t \mathbf{D} \nabla F} \tag{4.49}$$

Sometimes it is desirable to impose a different learning rate for the correct model vs. competing models. Therefore re-estimation Eq. (4.39) can be generalized to the following form [19, 48]:

$$\mathbf{\Phi}^{t+1} = \mathbf{\Phi}^t - \varepsilon_t U_t \nabla F(\mathbf{\Phi})\,|_{\mathbf{\Phi}=\mathbf{\Phi}^t} \tag{4.50}$$

where U_t is the learning bias matrix which is a positive definite matrix. One particular choice of U_t is $\mathbf{D}^{-1}$, where **D** is the Hessian matrix defined in Eq. (4.47). When the learning rate is set to be 1.0, Eq. (4.50) becomes Newton's algorithm, where the gradient descent is chosen to minimize the second-order expansion. Equation (4.50) becomes:

$$\mathbf{\Phi}^{t+1} = \mathbf{\Phi}^t - \mathbf{D}^{-1} \nabla F(\mathbf{\Phi})\,|_{\mathbf{\Phi}=\mathbf{\Phi}^t} \tag{4.51}$$

When probabilistic parameters are iteratively re-estimated, probabilistic constraints must be satisfied in each iteration as probability measure, such as:

1. For discrete distributions, all the values of the probability function ought to be nonnegative. Moreover the sum of all discrete probability values needs to be one, i.e., $\sum_i a_i = 1$
2. For continuous distributions (assuming Gaussian density family), the variance needs to be nonnegative. For Gaussian mixtures, the diagonal covariance entries need to be nonnegative and the sum of mixture weights needs to be one, i.e., $\sum_i c_i = 1$

In general, gradient descent is an unconstrained minimization (or maximization) process that needs to be modified to accommodate constrained minimization (or maximization) problems. The tricks to use are parameter transformations that implicitly maintain these constraints during gradient descent. The original parameters are updated through the inverse transform from the transformed parameter space to the original parameter space. The transformation is done in such a way that constraints on the original parameter are always maintained. Some of these transformations are listed as follows [48]:

1. For probabilities which need to be nonnegative and sum to one, like discrete probability function and mixture weight, the following transformation can be performed:

$$a_i = \frac{\exp(\tilde{a}_i)}{\sum_k \exp(\tilde{a}_k)} \tag{4.52}$$

2. For mean μ and variance (or diagonal covariance entries) σ^2, the following transformation can be used.

$$\mu = \tilde{\mu}\sigma \tag{4.53}$$

$$\sigma = \exp(\tilde{\sigma}) \tag{4.54}$$

After the transformations, we can now compute the gradient with respect to the transformed parameters $(\tilde{a}_i, \tilde{\mu}, \tilde{\sigma})$ using the chain rules. Once the new estimate for the transformed parameters is obtained through gradient descent, one can easily transform them back to the original parameter domain.

4.3.2. Minimum-Error-Rate Estimation

Parameter estimation techniques described so far aim to maximize either the likelihood (class-conditional probability) (MLE and MAP) or the posterior probability (MMIE) in Bayes' equation, Eq. (4.1). Although the criteria used in those estimation methods all have their own merit and under some conditions should lead to satisfactory results, the ultimate parameter estimation criterion for pattern recognition should be made to minimize the recognition error rate (or the Bayes' risk) directly. *Minimum-error-rate estimation* is also called *minimum-classification-error* (MCE) training, or discriminative training. Similar to MMIE, the algorithm generally tests the classifier using re-estimated models in the training procedure, and subsequently improves the correct models and suppresses mis-recognized or near-miss models.[9] Neural networks are in this class. Although minimum-error-rate estimation cannot be easily applied, it is still attractive that the criterion is identical to the goal of the spoken language systems.

We have used the posterior probability $p(\omega_i \mid \mathbf{x})$ in Bayes' rule as the discriminant function. In fact, just about any discriminant function can be used for minimum-error-rate estimation. For example, as described in Section 4.1.2, a Bayes' Gaussian classifier is equivalent to a quadratic discriminant function. The goal now is to find the estimation of parameters for a discriminant function family $\{d_i(\mathbf{x})\}$ to achieve the minimum error rate. One such error measure is defined in Eq. (4.5). The difficulty associated with the discriminative training approach lies in the fact that the error function needs to be consistent with the true error rate measure and also suitable for optimization.[10] Unfortunately, the error function defined in Section 4.1.1 [Eq. (4.5)] is based on a finite set, which is a piecewise constant function of the parameter vector $\mathbf{\Phi}$. It is not suitable for optimization.

To find an alternative smooth error function for MCE, let us assume that the discriminant function family contains *s* discriminant functions $d_i(\mathbf{x}, \mathbf{\Phi})$, *i* = 1, 2,…, s. $\mathbf{\Phi}$ denotes

[9] A near-miss model is the incorrect model that has highest likelihood other than that of the correct model.
[10] In general, a function is optimizable if it is a smooth function and has a derivative.

the entire parameter set for *s* discriminant functions. We also assume that all the discriminant functions are nonnegative. We define the following error (misclassification) measure:

$$e_i(\mathbf{x}) = -d_i(\mathbf{x}, \mathbf{\Phi}) + \left[\frac{1}{s-1} \sum_{j \neq i} d_j(\mathbf{x}, \mathbf{\Phi})^{\eta} \right]^{1/\eta} \tag{4.55}$$

where η is a positive number. The intuition behind the above measure is the attempt to enumerate the decision rule. For a ω_i class input **x**, $e_i(\mathbf{x}) > 0$ implies recognition error; while $e_i(\mathbf{x}) \leq 0$ implies correct recognition. The number η can be thought to be a coefficient to select competing classes in Eq. (4.55). When $\eta = 1$, the competing class term is the average of all the competing discriminant function scores. When $\eta \to \infty$, the competing class term becomes $\max_{j \neq i} d_j(\mathbf{x}, \mathbf{\Phi})$ representing the discriminant function score for the top competing class. By varying the value of η, one can take all the competing classes into account based on their individual significance.

To transform $e_i(\mathbf{x})$ into a normalized smooth function, we can use the sigmoid function to embed $e_i(\mathbf{x})$ in a smooth zero-one function. The loss function can be defined as follows:

$$l_i(\mathbf{x}; \mathbf{\Phi}) = sigmoid(e_i(\mathbf{x})) \tag{4.56}$$

where $sigmoid(x) = \dfrac{1}{1+e^{-x}}$ (4.57)

When $e_i(\mathbf{x})$ is a big negative number, which indicates correct recognition, the loss function $l_i(\mathbf{x}; \mathbf{\Phi})$ has a value close to zero, which implies no loss incurred. On the other hand, when $e_i(\mathbf{x})$ is a positive number, it leads to a value between zero and one that indicates the likelihood of an error. Thus $l_i(\mathbf{x}; \mathbf{\Phi})$ essentially represents a soft recognition error count.

For any data sample **x**, the recognizer's loss function can be defined as:

$$l(\mathbf{x}, \mathbf{\Phi}) = \sum_{i=1}^{s} l_i(\mathbf{x}, \mathbf{\Phi}) \delta(\omega = \omega_i) \tag{4.58}$$

where $\delta(\bullet)$ is a Boolean function which will return 1 if the argument is true and 0 if the argument is false. Since **X** is a random vector, the expected loss according to Eq. (4.58) can be defined as:

$$L(\mathbf{\Phi}) = E_{\mathbf{X}}(l(\mathbf{x}, \mathbf{\Phi})) = \sum_{i=1}^{s} \int_{\omega=\omega_i} l(\mathbf{x}, \mathbf{\Phi}) p(\mathbf{x}) d\mathbf{x} \tag{4.59}$$

Since $\max_{\mathbf{\Phi}}\left[\int f(\mathbf{x},\mathbf{\Phi})d\mathbf{x}\right] = \int\left[\max_{\mathbf{\Phi}} f(\mathbf{x},\mathbf{\Phi})\right]d\mathbf{x}$, $\mathbf{\Phi}$ can be estimated by gradient descent over $l(\mathbf{x},\mathbf{\Phi})$ instead of expected loss $L(\mathbf{\Phi})$. That is, minimum classification error training of parameter $\mathbf{\Phi}$ can be estimated by first choosing an initial estimate $\mathbf{\Phi}_0$ and the following iterative estimation equation:

$$\mathbf{\Phi}^{t+1} = \mathbf{\Phi}^t - \varepsilon_t \nabla l(\mathbf{x},\mathbf{\Phi})\,|_{\mathbf{\Phi}=\mathbf{\Phi}^t} \tag{4.60}$$

You can follow the gradient descent procedure described in Section 4.3.1.1 to achieve the MCE estimate of $\mathbf{\Phi}$.

Both MMIE and MCE are much more computationally intensive than MLE, owing to the inefficiency of gradient descent algorithms. Therefore, discriminant estimation methods, like MMIE and MCE, are usually used for tasks containing few classes or data samples. A more pragmatic approach is *corrective training* [6], which is based on a very simple error-correcting procedure. First, a labeled training set is used to train the parameters for each corresponding class by standard MLE. For each training sample, a list of confusable classes is created by running the recognizer and kept as its *near-miss* list. Then, the parameters of the correct class are moved in the direction of the data sample, while the parameters of the "near-miss" class are moved in the opposite direction of the data samples. After all training samples have been processed; the parameters of all classes are updated. This procedure is repeated until the parameters for all classes converge. Although there is no theoretical proof that such a process converges, some experimental results show that it outperforms both MLE and MMIE methods [4].

We have described various estimators: minimum mean square estimator, maximum likelihood estimator, maximum posterior estimator, maximum mutual information estimator, and minimum error estimator. Although based on different training criteria, they are all powerful estimators for various pattern recognition problems. Every estimator has its strengths and weaknesses. It is almost impossible always to favor one over the others. Instead, you should study their characteristics and assumptions and select the most suitable ones for the domains you are working on.

In the following section we discuss *neural networks*. Both neural networks and MCE estimations follow a very similar discriminant training framework.

4.3.3. Neural Networks

In the area of pattern recognition, the advent of new learning procedures and the availability of high-speed parallel supercomputers have given rise to a renewed interest in neural networks.[11] Neural networks are particularly interesting for speech recognition, which requires massive constraint satisfaction, i.e., the parallel evaluation of many clues and facts and their interpretation in the light of numerous interrelated constraints. The computational flexibility of the human brain comes from its large number of neurons in a mesh of axons and dendrites. The communication between neurons is via the synapse and afferent fibers. There are

[11] A neural network is sometimes called an artificial neural network (ANN), a neural net, or a connectionist model.

many billions of neural connections in the human brain. At a simple level it can be considered that nerve impulses are comparable to the phonemes of speech, or to letters, in that they do not themselves convey meaning but indicate different intensities [95, 101] that are interpreted as meaningful units by *the language of the brain*. Neural networks attempt to achieve real-time response and human like performance using many simple processing elements operating in parallel as in biological nervous systems. Models of neural networks use a particular topology for the interactions and interrelations of the connections of the *neural units*. In this section we describe the basics of neural networks, including the multi-layer perceptrons and the back-propagation algorithm for training neural networks.

4.3.3.1. Single-Layer Perceptrons

Figure 4.10 shows a basic *single-layer perceptron*. Assuming there are N inputs, labeled as $x_1, x_2, \ldots, x_N$, we can form a linear function with weights $w_{0j}, w_{1j}, w_{2j}, \ldots, w_{Nj}$ to give the output y_j, defined as

$$y_j = w_0 + \sum_{i=1}^{N} w_{ij} x_i = \mathbf{w}_j \mathbf{x}^t = g_j(\mathbf{x}) \tag{4.61}$$

where $\mathbf{w}_j = (w_{0j}, w_{1j}, w_{2j}, \ldots, w_{Nj})$ and $\mathbf{x} = (1, x_1, x_2, \ldots, x_N)$.

For pattern recognition purposes, we associate each class ω_j out of s classes $(\omega_1, \omega_2, \ldots, \omega_s)$ with such a linear discriminant function $g_j(\mathbf{x})$. By collecting all the discriminant functions, we will have the following matrix representation:

$$\mathbf{y} = \mathbf{g}(\mathbf{x}) = \mathbf{W}^t \mathbf{x} \tag{4.62}$$

where $\mathbf{g}(\mathbf{x}) = (g_1(\mathbf{x}), g_2(\mathbf{x}), \ldots, g_s(\mathbf{x}))^t$; $\mathbf{W} = (\mathbf{w}_1^t, \mathbf{w}_2^t, \ldots, \mathbf{w}_s^t)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_s)^t$. The pattern recognition decision can then be based on these discriminant functions as in Bayes' decision theory. That is,

$$\mathbf{x} \in \omega_k \quad \text{iff} \quad k = \arg\max_i g_i(\mathbf{x}) \tag{4.63}$$

The *perceptron training algorithm* [68], guaranteed to converge for linearly separable classes, is often used for training the weight matrix $\mathbf{W}$. The algorithm basically divides the sample space $\Re^N$ into regions of corresponding classes. The decision boundary is characterized by hyper-planes of the following form:

$$g_i(\mathbf{x}) - g_j(\mathbf{x}) = 0 \quad \forall i \neq j \tag{4.64}$$

Using linear discriminant functions is equivalent to estimating Bayes' Gaussian densities in which all densities are sharing the same covariance matrix. Unfortunately, for data samples that are not linearly separable, the perceptron algorithm does not converge. However, if we can relax the definition of classification errors in this case, we can still use a powerful algorithm to train the weight matrix $\mathbf{W}$. This approach is the *Least Square Error*

(LSE) algorithm described in Chapter 3, which aims at minimizing *sum-of-squared-error* (SSE) criterion, instead of minimizing the classification errors. The sum-of-squared-error is defined as:

$$\text{SSE} = \sum_i \sum_{\mathbf{x}\in\omega_i} \| \mathbf{g}(\mathbf{x}) - \delta_i \|^2 \tag{4.65}$$

where δ_i is an *M*-dimensional *index vector* with all zero components except that the i^{th} one is 1.0, since the desired output for $\mathbf{g}(\mathbf{x})$ is typically equal to 1.0 if $\mathbf{x} \in \omega_i$ and 0 if $\mathbf{x} \notin \omega_i$.

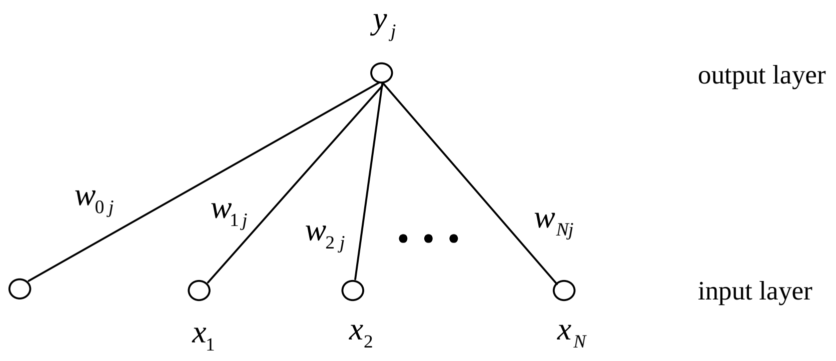

Figure 4.10 A single-layer perceptron.

The use of LSE leads to discriminant functions that have real outputs approximating the values 1 or 0. Suppose there are *S* input vectors $\mathbf{X} = (\mathbf{x}_1^t, \mathbf{x}_2^t, \ldots, \mathbf{x}_S^t)$ in the training set. Similar to the LSE for linear functions described in Chapter 3 (cf. Section 3.2.1.2), the LSE estimate of weight matrix $\mathbf{W}$ will have the following closed form:

$$\mathbf{W} = ((\mathbf{X}\mathbf{X}^t))^{-1}\mathbf{L}\Sigma \tag{4.66}$$

where $\mathbf{L}$ is a $(N+1)\times s$ matrix where the *k*-th column is the mean vector $\boldsymbol{\mu}_k = (1, \mu_{k1}, \mu_{k2}, \ldots, \mu_{kN})^t$ of all the vectors classified into class ω_k , and Σ is an $s \times s$ diagonal matrix with diagonal entry $c_{k,k}$ representing the number of vectors classified into ω_k . LSE estimation using linear discriminant functions is equivalent to estimating Bayes' Gaussian densities where all the densities are assumed to share the same covariance matrix [98].

Although the use of LSE algorithm solves the convergence problems, it loses the power of nonlinear logical decision (i.e., minimizing the classification error rate), since it is only approximating the simple logical decision between alternatives. An alternative approach is to use a smooth and differential sigmoid function as the threshold function:

$$\begin{aligned}\mathbf{y} &= sigmoid(\mathbf{g}(\mathbf{x})) = sigmoid((g_1(\mathbf{x}), g_2(\mathbf{x}), \ldots, g_s(\mathbf{x}))^t) \\ &= (sigmoid(g_1(\mathbf{x})), sigmoid(g_2(\mathbf{x})), \ldots, sigmoid(g_s(\mathbf{x})))^t\end{aligned} \tag{4.67}$$

where $sigmoid(x)$ is the sigmoid function defined in Eq. (4.57). With the sigmoid function, the following new sum-of-squared-error term closely tracks the classification error:

$$\text{NSSE} = \sum_i \sum_{\mathbf{x}\in\omega_i} \| sigmoid(\mathbf{g}(\mathbf{x})) - \delta_i \|^2 \tag{4.68}$$

where δ_i is the same index vector defined above. Since there is no analytic way of minimizing a nonlinear function, the use of the sigmoid threshold function requires an iterative gradient descent algorithm, back-propagation, which will be described in the next section.

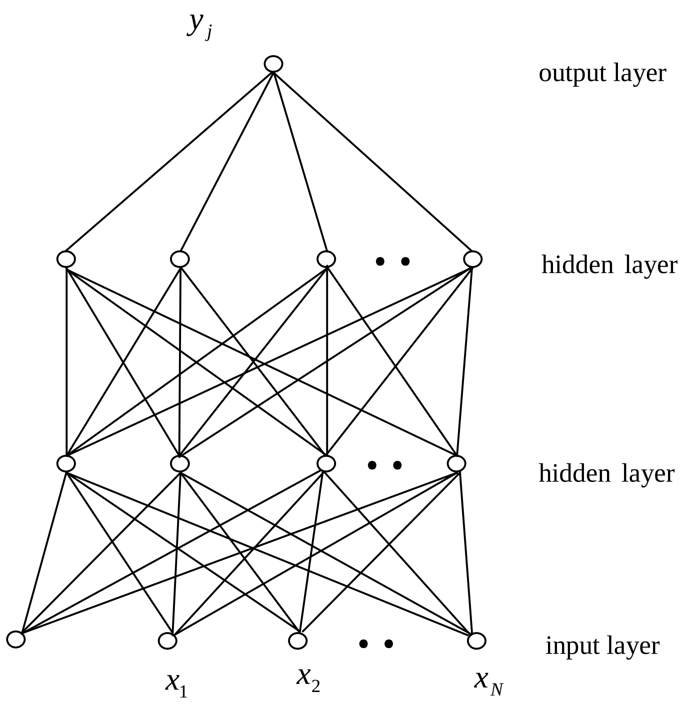

Figure 4.11 A multi-layer perceptron with four total layers. The middle two layers are hidden.

4.3.3.2. Multi-Layer Perceptron

One of the technical developments sparking the recent resurgence of interest in neural networks has been the popularization of *multi-layer perceptrons* (MLP) [37, 90]. Figure 4.11 shows a multi-layer perceptron. In contrast to a single-layer perceptron, it has two hidden layers. The hidden layers can be viewed as feature extractors. Each layer has the same computation models as the single-layer perceptron; i.e., the value of each node is computed as a linear weighted sum of the input nodes and passed to a sigmoid type of threshold function.

$$\begin{aligned}\mathbf{h}_1 &= sigmoid(\mathbf{g}_{h1}(\mathbf{x})) = sigmoid(\mathbf{W}^t_{h1}\mathbf{x})\\ \mathbf{h}_2 &= sigmoid(\mathbf{g}_{h2}(\mathbf{h}_1)) = sigmoid(\mathbf{W}^t_{h2}\mathbf{h}_1)\\ \mathbf{y} &= sigmoid(\mathbf{g}_y(\mathbf{h}_2)) = sigmoid(\mathbf{W}^t_y\mathbf{h}_2)\end{aligned} \tag{4.69}$$

where $sigmoid(x)$ is the sigmoid function defined in Eq. (4.57).

According to Eq. (4.69), we can propagate the computation from input layer to output layer and denote the output layer as a nonlinear function of the input layer.

$$\mathbf{Y} = MLP(\mathbf{x}) \tag{4.70}$$

Let's denote $O(\mathbf{x})$ as the desired output for input vector $\mathbf{x}$. For pattern classification, $O(\mathbf{x})$ will be an s-dimensional vector with the desired output pattern set to one and the remaining patterns set to zero. As we mentioned before, there is no analytic way to minimize the mean square error $E = \sum_{\mathbf{x}} \| MLP(\mathbf{x}) - O(\mathbf{x}) \|^2$. Instead, an iterative gradient descent algorithm called back propagation [89, 90] needs to be used to reduce to error. Without loss of generality, we assume there is only one input vector $\mathbf{x} = (1, x_1, x_2, \ldots, x_N)$ with desired output $\mathbf{o} = (o_1, o_2, \ldots, o_s)$. All the layers in the MLP are numbered 0, 1, 2,… upward from the input layer. The back propagation algorithm can then be described as in Algorithm 4.1.

In computing the partial derivative $\frac{\partial E}{\partial w^k_{ij}(t)}$, you need to use the chain rule. w^K_{ij} is the weight connecting the output layer and the last hidden layer; the partial derivative is:

$$\begin{aligned}\frac{\partial E}{\partial w^K_{ij}} &= \frac{\partial(\sum_{i=1}^{s}(y_i - o_i)^2)}{\partial w^K_{ij}}\\ &= \frac{\partial(\sum_{i=1}^{s}(y_i - o_i)^2)}{\partial y_j} \times \frac{\partial y_j}{\partial(w^K_{0j} + \sum_{i=1}^{N} w^K_{ij} v^{K-1}_i)} \times \frac{\partial(w^K_{0j} + \sum_{i=1}^{N} w^K_{ij} v^{K-1}_i)}{\partial w^K_{ij}}\\ &= 2(y_i - o_i) y_j (y_i - 1) v^{K-1}_i\end{aligned} \tag{4.71}$$

For layers $k = K-1, K-2, \cdots$, one can apply chain rules similarly for gradient $\frac{\partial E}{\partial w^k_{ij}(t)}$.

The back propagation algorithm is a generalization of the minimum mean squared error (MMSE) algorithm. It uses a gradient search to minimize the difference between the desired outputs and the actual net outputs, where the optimized criterion is directly related to pattern classification. With initial parameters for the weights, the training procedure is then repeated to update the weights until the cost function is reduced to an acceptable value or remains unchanged. In the algorithm described above, we assume a single training example. In real-world application, these weights are estimated from a large number of training observations in a manner similar to hidden Markov modeling. The weight updates in the Step 3 are accumulated over all the training data. The actual gradient is then estimated for the complete set of training data before the starting of the next iteration. Note that the estimation criterion for neural networks is directly related to classification rather than the maximum likelihood.

ALGORITHM 4.1 *THE BACK PROPAGATION ALGORITHM*

Step 1: Initialization: Set $t = 0$ and choose initial weight matrices $\mathbf{W}$ for each layer. Let's denote $w_{ij}^k(t)$ as the weighting coefficients connecting i^{th} input node in layer $k-1$ and j^{th} output node in layer k at time t.

Step 2: Forward Propagation: Compute the values in each node from input layer to output layer in a propagating fashion, for $k = 1$ to K

$$v_j^k = sigmoid(w_{0j}(t) + \sum_{i=1}^{N} w_{ij}^k(t) v_i^{k-1}) \quad \forall j \tag{4.72}$$

where $sigmoid(x) = \dfrac{1}{1+e^{-x}}$ and v_j^k is denotes as the j^{th} node in the k^{th} layer

Step 3: Back Propagation: Update the weights matrix for each layer from output layer to input layer according to:

$$\overline{w}_{ij}^k(t+1) = w_{ij}^k(t) - \alpha \frac{\partial E}{\partial w_{ij}^k(t)} \tag{4.73}$$

where $E = \sum_{i=1}^{s} \| y_i - o_i \|^2$ and $(y_1, y_2, \dots y_s)$ is the computed output vector in Step 2.

α is referred to as the learning rate and has to be small enough to guarantee convergence. One popular choice is $1/(t+1)$.

Step 4: Iteration: Let $t = t + 1$ Repeat Steps 2 and 3 until some convergence condition is met.

4.4. UNSUPERVISED ESTIMATION METHODS

As described in Section 4.2, in unsupervised learning, information about class ω of the data sample **x** is unavailable. Data observed are *incomplete* since the class data ω is missing. One might wonder why we are interested in such an unpromising problem, and whether or not it is possible to learn anything from incomplete data. Interestingly enough, the formal solution to this problem is almost identical to the solution for the supervised learning case – MLE. We discuss vector quantization (VQ), which uses principles similar to the EM algorithm. It is important in its own right in spoken language systems.

4.4.1. Vector Quantization

As described in Chapter 3, source coding refers to techniques that convert the signal source into a sequence of bits that are transmitted over a communication channel and then used to reproduce the original signal at a different location or time. In speech communication, the reproduced sound usually allows some acceptable level of distortion to achieve low bit rate. The goal of source coding is to reduce the number of bits necessary to transmit or store data, subject to a distortion or fidelity criterion, or equivalently, to achieve the minimum possible distortion for a prescribed bit rate. *Vector quantization* (VQ) is one of the most efficient source-coding techniques

Quantization is the process of approximating continuous amplitude signals by discrete symbols. The quantization of a single signal value or parameter is referred to as scalar quantization. In contrast, joint quantization of multiple signal values or parameters is referred to as vector quantization. Conventional pattern recognition techniques have been used effectively to solve the quantization or data compression problem with successful application to speech coding, image coding, and speech recognition [36, 85]. In both speech recognition and synthesis systems, vector quantization serves an important role in many aspects of the systems, ranging from discrete acoustic prototypes of speech signals for the discrete HMM, to robust signal processing and data compression.

A vector quantizer is described by a codebook, which is a set of fixed *prototype vectors* or reproduction vectors. Each of these prototype vectors is also referred to as a codeword. To perform the quantization process, the input vector is matched against each codeword in the codebook using some *distortion measure*. The input vector is then replaced by the index of the codeword with the smallest distortion. Therefore, a description of the vector quantization process includes:

1. the distortion measure;
2. the generation of each codeword in the codebook.

4.4.1.1. Distortion Measures

Since vectors are replaced by the index of the codeword with smallest distortion, the transmitted data can be recovered only by replacing the code index sequence with the corresponding codeword sequence. This inevitably causes distortion between the original data and the transmitted data. How to minimize the distortion is thus the central goal of vector quantization. This section describes a couple of the most common distortion measures.

Assume that $\mathbf{x} = (x_1, x_2, \ldots, x_d)^t \in R^d$ is a d-dimensional vector whose components $\{x_k, 1 \le k \le d\}$ are real-valued, continuous-amplitude random variables. After vector quantization, the vector **x** is mapped (quantized) to another discrete-amplitude *d*-dimensional vector **z**.

$$\mathbf{z} = q(\mathbf{x}) \tag{4.74}$$

In Eq. (4.74) *q*() is the quantization operator. Typically, **z** is a vector from a finite set $\mathbf{Z} = \{\mathbf{z}_j, 1 \le j \le M\}$, where $\mathbf{z}_j$ is also a *d*-dimensional vector. The set **Z** is referred to as the codebook, *M* is the size of the codebook, and $\mathbf{z}_j$ is *j*th *codeword*. The size *M* of the codebook is also called the number of partitions (or levels) in the codebook. To design a codebook, the *d*-dimensional space of the original random vector **x** can be partitioned into *M* regions or cells $\{C_i, 1 \le i \le M\}$, and each cell C_i is associated with a codeword vector $\mathbf{z}_i$. VQ then maps (quantizes) the input vector **x** to codeword $\mathbf{z}_i$ if **x** lies in C_i. That is,

$$q(\mathbf{x}) = \mathbf{z}_i \text{ if } \mathbf{x} \in C_i \tag{4.75}$$

An example of partitioning of a two-dimensional space (*d* = 2) for the purpose of vector quantization is shown in Figure 4.12. The shaded region enclosed by the dashed lines is the cell C_i. Any input vector **x** that lies in the cell C_i is quantized as $\mathbf{z}_i$. The shapes of the various cells can be different. The positions of the codewords within each cell are shown by dots in Figure 4.12. The codeword $\mathbf{z}_i$ is also referred to as the *centroid* of the cell C_i because it can be viewed as the central point of the cell C_i.

When **x** is quantized as **z**, a quantization error results. A distortion measure *d*(**x**, **z**) can be defined between **x** and **z** to measure the quantization quality. Using this distortion measure, Eq. (4.75) can be reformulated as follows:

$$q(\mathbf{x}) = \mathbf{z}_i \text{ if and only if } i = \underset{k}{\operatorname{argmin}}\ d(\mathbf{x}, \mathbf{z}_k) \tag{4.76}$$

The distortion measure between **x** and **z** is also known as a distance measure in the speech context. The measure must be tractable in order to be computed and analyzed, and also must be subjectively relevant so that differences in distortion values can be used to indicate differences in original and transmitted signals. The most commonly used measure is the Euclidean distortion measure, which assumes that the distortions contributed by quantizing the different parameters are equal. Therefore, the distortion measure *d*(**x**, **z**) can be defined as follows:

$$d(\mathbf{x}, \mathbf{z}) = (\mathbf{x} - \mathbf{z})^t (\mathbf{x} - \mathbf{z}) = \sum_{i=1}^{d} (x_i - z_i)^2 \tag{4.77}$$

The distortion defined in Eq. (4.77) is also known as sum of squared error. In general, unequal weights can be introduced to weight certain contributions to the distortion more than others. One choice for weights that is popular in many practical applications is to use the inverse of the covariance matrix of **z**.

$$d(\mathbf{x}, \mathbf{z}) = (\mathbf{x} - \mathbf{z})^t \Sigma^{-1} (\mathbf{x} - \mathbf{z}) \tag{4.78}$$

This distortion measure, known as the *Mahalanobis* distance, is actually the exponential term in a Gaussian density function.

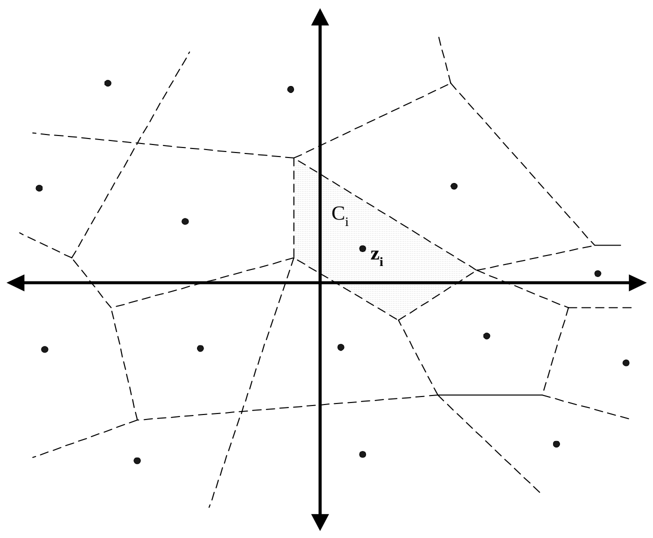

Figure 4.12 Partitioning of a two-dimensional space into 16 cells.

Another way to weight the contributions to the distortion measure is to use *perceptually*-based distortion measures. Such distortion measures take advantage of subjective judgments of perceptual difference caused by two different signals. A perceptually-based distortion measure has the property that signal changes that make the sounds being perceived different should be associated with large distances. Similarly signal changes that keep the sound perceived the same should be associated with small distances. A number of perceptually based distortion measures have been used in speech coding [3, 75, 76].

4.4.1.2. The *K*-Means Algorithm

To design an *M*-level codebook, it is necessary to partition *d*-dimensional space into *M* cells and associate a quantized vector with each cell. Based on the source-coding principle, the criterion for optimization of the vector quantizer is to minimize overall average distortion over all *M*-levels of the VQ. The overall average distortion can be defined by

$$\begin{aligned} D = E\left[d(\mathbf{x},\mathbf{z})\right] &= \sum_{i=1}^{M} p(\mathbf{x}\in C_i)E\left[d(\mathbf{x},\mathbf{z}_i)\,|\,\mathbf{x}\in C_i\right] \\ &= \sum_{i=1}^{M} p(\mathbf{x}\in C_i)\int_{x\in C_i} d(\mathbf{x},\mathbf{z}_i)\,p(\mathbf{x}\,|\,\mathbf{x}\in C_i)d\mathbf{x} = \sum_{i=1}^{M} D_i \end{aligned} \tag{4.79}$$

where the integral is taken over all components of vector **x**; $p(\mathbf{x} \in C_i)$ denotes the prior probability of codeword $\mathbf{z}_i$; $p(\mathbf{x} | \mathbf{x} \in C_i)$ denotes the multidimensional probability density function of **x** in cell C_i; and D_i is the average distortion in cell C_i. No analytic solution exists to guarantee global minimization of the average distortion measure for a given set of speech data. However, an iterative algorithm, which guarantees a local minimum, exists and works well in practice.

We say a quantizer is optimal if the overall average distortion is minimized over all *M*-levels of the quantizer. There are two necessary conditions for optimality. The first is that the optimal quantizer is realized by using a nearest-neighbor selection rule as specified by Eq. (4.76). Note that the average distortion for each cell C_i

$$E\left[d(\mathbf{x}, \mathbf{z}_i) \mid \mathbf{x} \in C_i\right] \tag{4.80}$$

can be minimized when $\mathbf{z}_i$ is selected such that $d(\mathbf{x}, \mathbf{z}_i)$ is minimized for **x**. This means that the quantizer must choose the codeword that results in the minimum distortion with respect to **x**. The second condition for optimality is that each codeword $\mathbf{z}_i$ is chosen to minimize the average distortion in cell C_i. That is, $\mathbf{z}_i$ is the vector that minimizes

$$D_i = p(\mathbf{z}_i) E\left[d(\mathbf{x}, \mathbf{z}) | \mathbf{x} \in C_i\right] \tag{4.81}$$

Since the overall average distortion *D* is a linear combination of average distortions in C_i, they can be independently computed after classification of **x**. The vector $\mathbf{z}_i$ is called the centroid of the cell C_i and is written

$$\mathbf{z}_i = \text{cent}(C_i) \tag{4.82}$$

The centroid for a particular region (cell) depends on the definition of the distortion measure. In practice, given a set of training vectors $\{\mathbf{x}_t, 1 \le t \le T\}$, a subset of K_i vectors will be located in cell C_i. In this case, $p(\mathbf{x} | \mathbf{z}_i)$ can be assumed to be $1/K_i$, and $p(\mathbf{z}_i)$ becomes K_i / T. The average distortion D_i in cell C_i can then be given by

$$D_i = \frac{1}{T} \sum_{\mathbf{x} \in C_i} d(\mathbf{x}, \mathbf{z}_i) \tag{4.83}$$

The second condition for optimality can then be rewritten as follows:

$$\hat{\mathbf{z}}_i = \underset{\mathbf{z}_i}{\arg\min} D_i(\mathbf{z}_i) = \underset{\mathbf{z}_i}{\arg\min} \frac{1}{T} \sum_{\mathbf{x} \in C_i} d(\mathbf{x}, \mathbf{z}_i) \tag{4.84}$$

When the sum of squared error in Eq. (4.77) is used for the distortion measure, the attempt to find such $\hat{\mathbf{z}}_i$ to minimize the sum of squared error is equivalent to least squared error estimation, which was described in Chapter 3. Minimization of D_i in Eq. (4.84) with respect to $\mathbf{z}_i$ is given by setting the derivative of D_i to zero:

$$\begin{aligned}\nabla_{\mathbf{z}_i} D_i &= \nabla_{\mathbf{z}_i} \frac{1}{T}\sum_{\mathbf{x}\in C_i}(\mathbf{x}-\mathbf{z}_i)^t(\mathbf{x}-\mathbf{z}_i)\\ &= \frac{1}{T}\sum_{\mathbf{x}\in C_i}\nabla_{\mathbf{z}_i}(\mathbf{x}-\mathbf{z}_i)^t(\mathbf{x}-\mathbf{z}_i)\\ &= \frac{-2}{T}\sum_{\mathbf{x}\in C_i}(\mathbf{x}-\mathbf{z}_i)=0\end{aligned} \tag{4.85}$$

By solving Eq. (4.85), we obtain the least square error estimate of centroid $\hat{\mathbf{z}}_i$ simply as the sample mean of all the training vectors **x**, quantized to cell C_i:

$$\hat{\mathbf{z}}_i = \frac{1}{K_i}\sum_{\mathbf{x}\in C_i}\mathbf{x} \tag{4.86}$$

If the Mahalanobis distance measure (Eq. (4.78)) is used, minimization of D_i in Eq. (4.84) can be done similarly:

$$\begin{aligned}\nabla_{\mathbf{z}_i} D_i &= \nabla_{\mathbf{z}_i} \frac{1}{T}\sum_{\mathbf{x}\in C_i}(\mathbf{x}-\mathbf{z}_i)^t\Sigma^{-1}(\mathbf{x}-\mathbf{z}_i)\\ &= \frac{1}{T}\sum_{\mathbf{x}\in C_i}\nabla_{\mathbf{z}_i}(\mathbf{x}-\mathbf{z}_i)^t\Sigma^{-1}(\mathbf{x}-\mathbf{z}_i)\\ &= \frac{-2}{T}\sum_{\mathbf{x}\in C_i}\Sigma^{-1}(\mathbf{x}-\mathbf{z}_i)=0\end{aligned} \tag{4.87}$$

and centroid $\hat{\mathbf{z}}_i$ is obtained from

$$\hat{\mathbf{z}}_i = \frac{1}{K_i}\sum_{\mathbf{x}\in C_i}\mathbf{x} \tag{4.88}$$

One can see that $\hat{\mathbf{z}}_i$ is again the sample mean of all the training vectors **x**, quantized to cell C_i. Although Eq. (4.88) is obtained based on the Mahalanobis distance measure, it also works with a large class of Euclidean-like distortion measures [61]. Since the Mahalanobis distance measure is actually the exponential term in a Gaussian density, minimization of the distance criterion can be easily translated into maximization of the logarithm of the Gaussian likelihood. Therefore, similar to the relationship between least square error estimation for the linear discrimination function and the Gaussian classifier described in Section 4.3.3.1, the distance minimization process (least square error estimation) above is in fact a *maximum likelihood estimation*.

According to these two conditions for VQ optimality, one can iteratively apply the nearest-neighbor selection rule and Eq. (4.88) to get the new centroid $\hat{\mathbf{z}}_i$ for each cell in order to minimize the average distortion measure. This procedure is known as the *k-means* algorithm or the *generalized Lloyd* algorithm [29, 34, 56]. In the *k*-means algorithm, the basic idea is to partition the set of training vectors into *M* clusters C_i $\left(1 \le i \le M\right)$ in such a way that the two necessary conditions for optimality described above are satisfied. The *k*-means algorithm can be described as follows:

ALGORITHM 4.2: *THE K-MEANS ALGORITHM*

Step 1: Initialization: Choose some adequate method to derive initial VQ codewords ($\mathbf{z}_{i,}\ 1 \le i \le M$) in the codebook.

Step 2: Nearest-neighbor Classification: Classify each training vector $\{\mathbf{x}_k\}$ into one of the cell s C_i by choosing the closest codeword $\mathbf{z}_i$ ($\mathbf{x} \in C_i$, iff $d(\mathbf{x}, \mathbf{z}_i) \le d(\mathbf{x}, \mathbf{z}_j)$ for all $j \ne i$). This classification is also called minimum-distance classifier.

Step 3: Codebook Updating: Update the codeword of every cell by computing the centroid of the training vectors in each cell according to Eq. (4.84) ($\hat{\mathbf{z}}_i = cent(C_i),\ 1 \le i \le M$).

Step 4: Iteration: Repeat steps 2 and 3 until the ratio of the new overall distortion *D* at the current iteration relative to the overall distortion at the previous iteration is above a preset threshold.

In the process of minimizing the average distortion measure, the *k*-means procedure actually breaks the minimization process into two steps. Assuming that the centroid $\mathbf{z}_i$ (or mean) for each cell C_i has been found, then the minimization process is found simply by partitioning all the training vectors into their corresponding cells according to the distortion measure. After all of the new partitions are obtained, the minimization process involves finding the new centroid within each cell to minimize its corresponding within-cell average distortion D_i based on Eq. (4.84). By iterating over these two steps, a new overall distortion *D* smaller than that of the previous step can be obtained.

Theoretically, the *k*-means algorithm can converge only to a local optimum [56]. Furthermore, any such solution is, in general, not unique [33]. Initialization is often critical to the quality of the eventual converged codebook. Global optimality may be approximated by repeating the *k*-means algorithm for several sets of codebook initialization values, and then one can choose the codebook that produces the minimum overall distortion. In the next subsection we will describe methods for finding a decent initial codebook.

4.4.1.3. The LBG Algorithm

Since the initial codebook is critical to the ultimate quality of the final codebook, it has been shown that it is advantageous to design an M-vector codebook in stages. This extended k-means algorithm is known as the LBG algorithm proposed by Linde, Buzo, and Gray [56]. The LBG algorithm first computes a 1-vector codebook, then uses a splitting algorithm on the codewords to obtain the initial 2-vector codebook, and continues the splitting process until the desired *M*-vector codebook is obtained. The procedure is formally implemented by the following algorithm:

ALGORITHM 4.3: *THE LBG ALGORITHM*

Step 1: Initialization: Set M (number of partitions or cells) =1. Find the centroid of all the training data according to Eq. (4.84).
Step 2: Splitting: Split M into $2M$ partitions by splitting each current codeword by finding two points that are far apart in each partition using a heuristic method, and use these two points as the new centroids for the new $2M$ codebook. Now set $M = 2M$.
Step 3: K-means Stage: Now use the k-means iterative algorithm described in the previous section to reach the best set of centroids for the new codebook.
Step 4: Termination: If M equals the VQ codebook size required, STOP; otherwise go to Step 2.

4.4.2. The EM Algorithm

We introduce the EM algorithm that is important to hidden Markov models and other learning techniques. It discovers model parameters by maximizing the log-likelihood of incomplete data and by iteratively maximizing the expectation of log-likelihood from complete data. The EM algorithm is a generalization of the VQ algorithm described above.

The EM algorithm can also be viewed as a generalization of the MLE method, when the data observed is incomplete. Without loss of generality, we use scale random variables here to describe the EM algorithm. Suppose we observe training data *y*. In order to determine the parameter vector Φ that maximizes $P(Y = y \mid \Phi)$, we would need to know some hidden data *x* (that is unobserved). For example, *x* may be a hidden number that refers to component densities of observable data *y*, or *x* may be the underlying hidden state sequence in *hidden Markov models* (as discussed in Chapter 8). Without knowing this hidden data *x*, we could not easily use the maximum likelihood estimation to estimate $\hat{\Phi}$, which maximizes $P(Y = y \mid \Phi)$. Instead, we assume a parameter vector Φ and estimate the probability that each *x* occurred in the generation of *y*. This way we can pretend that we had in fact observed a complete data pair (*x*, *y*), with frequency proportional to the probability $P(X = x, Y = y \mid \Phi)$, to compute a new $\bar{\Phi}$, the maximum likelihood estimate of Φ. We can then set the parameter vector Φ to be this new $\bar{\Phi}$ and repeat the process to iteratively improve our estimate.

The issue now is whether or not the process (EM algorithm) described above converges. Without loss of generality, we assume that both random variables *X* (unobserved) and *Y* (observed) are discrete random variables. According to Bayes rule,

$$P(X = x, Y = y \mid \overline{\Phi}) = P(X = x \mid Y = y, \overline{\Phi}) P(Y = y \mid \overline{\Phi}) \tag{4.89}$$

Our goal is to maximize the log-likelihood of the observable, real data *y* generated by parameter vector $\overline{\Phi}$. Based on Eq. (4.89), the log-likelihood can be expressed as follows:

$$\log P(Y = y \mid \overline{\Phi}) = \log P(X = x, Y = y \mid \overline{\Phi}) - \log P(X = x \mid Y = y, \overline{\Phi}) \tag{4.90}$$

Now, we take the conditional expectation of $\log P(Y = y \mid \overline{\Phi})$ over *X* computed with parameter vector Φ:

$$\begin{aligned} E_{\Phi}[\log P(Y = y \mid \overline{\Phi})]_{X \mid Y=y} &= \sum_x \left(P(X = x \mid Y = y, \Phi) \log P(Y = y \mid \overline{\Phi}) \right) \\ &= \log P(Y = y \mid \overline{\Phi}) \end{aligned} \tag{4.91}$$

where we denote $E_{\Phi}[f]_{X \mid Y=y}$ as the expectation of function *f* over *X* computed with parameter vector Φ. Then using Eq. (4.90) and (4.91) , the following expression is obtained:

$$\begin{aligned} \log P(Y = y \mid \overline{\Phi}) &= E_{\Phi}[\log P(X, Y = y \mid \overline{\Phi})]_{X \mid Y=y} - E_{\Phi}[\log P(X \mid Y = y, \overline{\Phi})]_{X \mid Y=y} \\ &= Q(\Phi, \overline{\Phi}) - H(\Phi, \overline{\Phi}) \end{aligned} \tag{4.92}$$

where

$$\begin{aligned} Q(\Phi, \overline{\Phi}) &= E_{\Phi}[\log P(X, Y = y \mid \overline{\Phi})]_{X \mid Y=y} \\ &= \sum_x \left(P(X = x \mid Y = y, \Phi) \log P(X = x, Y = y \mid \overline{\Phi}) \right) \end{aligned} \tag{4.93}$$

and

$$\begin{aligned} H(\Phi, \overline{\Phi}) &= E_{\Phi}[\log P(X \mid Y = y, \overline{\Phi})]_{X \mid Y=y} \\ &= \sum_x \left(P(X = x \mid Y = y, \overline{\Phi}) \log P(X = x \mid Y = y, \overline{\Phi}) \right) \end{aligned} \tag{4.94}$$

The convergence of the EM algorithm lies in the fact that if we choose $\overline{\Phi}$ so that

$$Q(\Phi, \overline{\Phi}) \ge Q(\Phi, \Phi) \tag{4.95}$$

then

$$\log P(Y = y \mid \overline{\Phi}) \ge \log P(Y = y \mid \Phi) \tag{4.96}$$

since it follows from Jensen's inequality that $H(\Phi, \overline{\Phi}) \le H(\Phi, \Phi)$ [21]. The function $Q(\Phi, \overline{\Phi})$ is known as the *Q*-function or auxiliary function. This fact implies that we can maximize the *Q*-function, which is the expectation of log-likelihood from complete data pair (*x*, *y*), to update parameter vector from Φ to $\overline{\Phi}$, so that the incomplete log-likelihood

$L(x,\Phi)$ increases monotonically. Eventually, the likelihood will converge to a local maximum if we iterate the process.

The name of the EM algorithm comes from E for expectation and M for maximization. The implementation of the EM algorithm includes the E (expectation) step, which calculates the auxiliary Q-function $Q(\Phi,\bar{\Phi})$ and the M (maximization) step, which maximizes $Q(\Phi,\bar{\Phi})$ over $\bar{\Phi}$ to obtain $\hat{\Phi}$. The general EM algorithm can be described in the following way.

ALGORITHM 4.4: *THE EM ALGORITHM*

Step 1: Initialization: Choose an initial estimate Φ.
Step 2: E-Step: Compute auxiliary Q-function $Q(\Phi,\bar{\Phi})$ (which is also the expectation of log-likelihood from complete data) based on Φ.
Step 3: M-Step: Compute $\hat{\Phi} = \arg\max_{\bar{\Phi}} Q(\Phi,\bar{\Phi})$ to maximize the auxiliary Q-function.
Step 4: Iteration: Set $\Phi = \hat{\Phi}$, repeat from Step 2 until convergence.

The M-step of the EM algorithm is actually a maximum likelihood estimation of complete data (assuming we know the unobserved data x based on observed data y and initial parameter vector Φ). The EM algorithm is usually used in applications where no analytic solution exists for maximization of log-likelihood of incomplete data. Instead, the Q-function is iteratively maximized to obtain the estimation of parameter vector.

4.4.3. Multivariate Gaussian Mixture Density Estimation

The vector quantization process described in Section 4.4.1 partitions the data space into separate regions based on some distance measure regardless of the probability distributions of original data. This process may introduce errors in partitions that could potentially destroy the original structure of data. An alternative way for modeling a VQ codebook is to use a family of Gaussian probability density functions, such that each cell will be represented by a (Gaussian) probability density function as shown in Figure 4.13. These probability density functions can then overlap, rather than partition, in order to represent the entire data space. The objective for a mixture Gaussian VQ is to maximize the likelihood of the observed data (represented by the product of the Gaussian mixture scores) instead of minimizing the overall distortion. The centroid of each cell (the mean vectors of each Gaussian pdf) obtained via such a representation may be quite different from that obtained using the traditional k-mean algorithm, since the distribution properties of the data are taken into account.

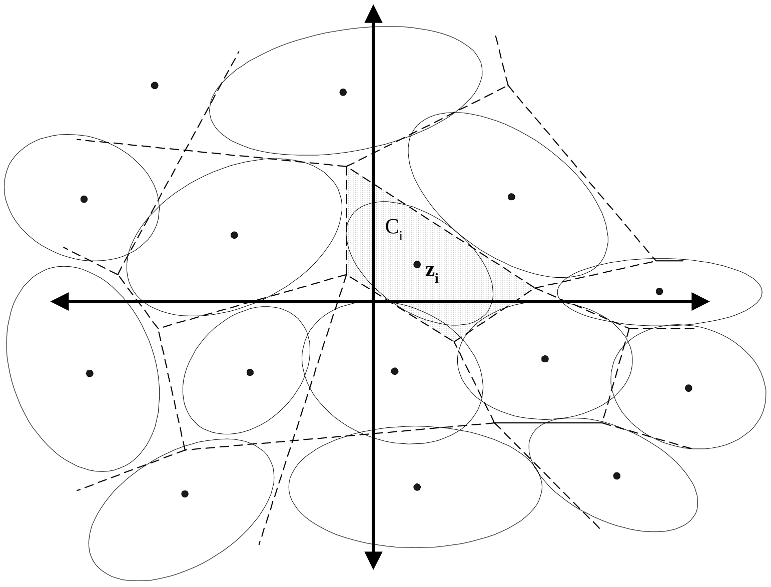

Figure 4.13 Partitioning of a two-dimensional space with 12 Gaussian density functions.

There should be an obvious analogy between the EM algorithm and the *k*-means algorithm described in the Section 4.4.1.2. In the *k*-means algorithm, the class information for the observed data samples is hidden and unobserved, so an EM-like algorithm instead of maximum likelihood estimate needs to be used. Therefore, instead of a single process of maximum likelihood estimation, the *k*-means algorithm first uses the old codebook to find the nearest neighbor for each data sample followed by maximum likelihood estimation of the new codebook and iterates the process until the distortion stabilizes. The steps 2 and 3 in the *k*-means algorithm are actually the E and M steps in the EM algorithm respectively.

Mixture density estimation [41] is a typical example of EM estimation. In the mixtures of Gaussian density, the probability density for observable data **y** is the weighted sum of each Gaussian component.

$$p(\mathbf{y} \mid \mathbf{\Phi}) = \sum_{k=1}^{K} c_k p_k(\mathbf{y} \mid \mathbf{\Phi}_k) = \sum_{k=1}^{K} c_k N_k(\mathbf{y} \mid \boldsymbol{\mu}_k, \mathbf{\Sigma}_k) \tag{4.97}$$

where $0 \le c_k \le 1$, for $1 \le k \le K$ and $\sum_{k=1}^{K} c_k = 1$.

Unlike the case of a single Gaussian estimation, we also need to estimate the mixture weight c_k. In order to do so, we can assume that observable data **y** come from one of the component densities $p_X(\mathbf{y} \mid \phi_X)$, where *X* is a random variable taking value from $\{1, 2, \ldots K\}$ to indicate the Gaussian component. It is clear that *x* is unobserved and used to specify the pdf component ϕ_X . Assuming that the probability density function for complete data (*x*,**y**) is given by the joint probability:

$$p(\mathbf{y},x\,|\,\mathbf{\Phi})=P(X=x)p_x(\mathbf{y}\,|\,\mathbf{\Phi}_x)=P(X=x)N_k(\mathbf{y}\,|\,\boldsymbol{\mu}_k,\mathbf{\Sigma}_k) \tag{4.98}$$

$P(X=x)$ can be regarded as the probability of the unobserved data *x* used to specify the component density $p_x(\mathbf{y}\,|\,\mathbf{\Phi}_x)$ from which the observed data **y** is drawn. If we assume the number of components is K and $\mathbf{\Phi}$ is the vector of all probability parameters ($P(X)$, $\mathbf{\Phi}_1,\mathbf{\Phi}_2,\ldots,\mathbf{\Phi}_K$), the probability density function of incomplete (observed) data **y** can be specified as the following marginal probability:

$$p(\mathbf{y}\,|\,\mathbf{\Phi})=\sum_x p(\mathbf{y},x\,|\,\mathbf{\Phi})=\sum_x P(X=x)p_x(\mathbf{y}\,|\,\mathbf{\Phi}_x) \tag{4.99}$$

By comparing Eq. (4.97) and (4.99), we can see that the mixture weight is represented as the probability function $P(X=x)$. That is,

$$c_k=P(X=k) \tag{4.100}$$

According to the EM algorithm, the maximization of the logarithm of the likelihood function $\log p(\mathbf{y}\,|\,\mathbf{\Phi})$ can be performed by iteratively maximizing the conditional expectation of the logarithm of Eq. (4.98), i.e., $\log p(\mathbf{y},x\,|\,\mathbf{\Phi})$. Suppose we have observed N independent samples: $\{\mathbf{y}_1,\mathbf{y}_2,\ldots,\mathbf{y}_N\}$ with hidden unobserved data $\{x_1,x_2,\ldots,x_N\}$; the *Q*-function can then be written as follows:

$$\begin{aligned}Q(\mathbf{\Phi},\bar{\mathbf{\Phi}})&=\sum_{i=1}^{N}Q_i(\mathbf{\Phi},\bar{\mathbf{\Phi}})=\sum_{i=1}^{N}\sum_{x_i}P(x_i\,|\,\mathbf{y}_i,\mathbf{\Phi})\log p(\mathbf{y}_i,x_i\,|\,\bar{\mathbf{\Phi}})\\&=\sum_{i=1}^{N}\sum_{x_i}\frac{p(\mathbf{y}_i,x_i\,|\,\mathbf{\Phi})}{p(\mathbf{y}_i\,|\,\mathbf{\Phi})}\log p(\mathbf{y}_i,x_i\,|\,\bar{\mathbf{\Phi}})\end{aligned} \tag{4.101}$$

By replacing items in Eq. (4.101) with Eqs. (4.98) and (4.100), the following equation can be obtained:

$$Q(\mathbf{\Phi},\bar{\mathbf{\Phi}})=\sum_{k=1}^{K}\gamma_k\log\bar{c}_k+\sum_{k=1}^{K}Q_\lambda(\mathbf{\Phi},\bar{\mathbf{\Phi}}_k) \tag{4.102}$$

where

$$\gamma_k^i=\frac{c_k p_k(\mathbf{y}_i\,|\,\mathbf{\Phi}_k)}{P(\mathbf{y}_i\,|\,\mathbf{\Phi})} \tag{4.103}$$

$$\gamma_k=\sum_{i=1}^{N}\gamma_k^i=\sum_{i=1}^{N}\frac{c_k p_k(\mathbf{y}_i\,|\,\mathbf{\Phi}_k)}{P(\mathbf{y}_i\,|\,\mathbf{\Phi})} \tag{4.104}$$

$$Q_\lambda(\Phi,\bar{\mathbf{\Phi}}_k) = \sum_{i=1}^{N} \gamma_k^i \log p_k(\mathbf{y}_i \mid \bar{\mathbf{\Phi}}_k) = \sum_{i=1}^{N} \frac{c_k p_k(\mathbf{y}_i \mid \mathbf{\Phi}_k)}{P(\mathbf{y}_i \mid \mathbf{\Phi})} \log p_k(\mathbf{y}_i \mid \bar{\mathbf{\Phi}}_k) \tag{4.105}$$

Now we can perform a maximum likelihood estimation on the complete data (x, $\mathbf{y}$) during the M-step. By taking the derivative with respect to each parameter and setting it to zero, we obtain the following EM re-estimate of c_k, m_k, and Σ_k:

$$\hat{c}_k = \frac{\gamma_k}{\sum_{k=1}^{K} \gamma_k} = \frac{\gamma_k}{N} \tag{4.106}$$

$$\hat{\boldsymbol{\mu}}_k = \frac{\sum_{i=1}^{N} \gamma_k^i \mathbf{y}_i}{\sum_{i=1}^{N} \gamma_k^i} = \frac{\sum_{i=1}^{N} \frac{c_k p_k(\mathbf{y}_i \mid \mathbf{\Phi}_k)\mathbf{y}_i}{P(\mathbf{y}_i \mid \mathbf{\Phi})}}{\sum_{i=1}^{N} \frac{c_k p_k(\mathbf{y}_i \mid \mathbf{\Phi}_k)}{P(\mathbf{y}_i \mid \mathbf{\Phi})}} \tag{4.107}$$

$$\hat{\Sigma}_k = \frac{\sum_{i=1}^{N} \gamma_k^i (\mathbf{y}_i - \boldsymbol{\mu}_k)(\mathbf{y}_i - \boldsymbol{\mu}_k)^t}{\sum_{i=1}^{N} \gamma_k^i} = \frac{\sum_{i=1}^{N} \frac{c_k p_k(\mathbf{y}_i \mid \mathbf{\Phi}_k)(\mathbf{y}_i - \boldsymbol{\mu}_k)(\mathbf{y}_i - \boldsymbol{\mu}_k)^t}{P(\mathbf{y}_i \mid \mathbf{\Phi})}}{\sum_{i=1}^{N} \frac{c_k p_k(\mathbf{y}_i \mid \mathbf{\Phi}_k)}{P(\mathbf{y}_i \mid \mathbf{\Phi})}} \tag{4.108}$$

The quantity γ_k^i defined in Eq. (4.103) can be interpreted as the posterior probability that the observed data $\mathbf{y}_i$ belong to Gaussian component k ($N_k(\mathbf{y} \mid \boldsymbol{\mu}_k, \Sigma_k)$). This information as to whether the observed data $\mathbf{y}_i$ should belong to Gaussian component k is hidden and can only be observed through the hidden variable x (c_k). The EM algorithm described above is used to uncover how likely the observed data $\mathbf{y}_i$ are expected to be in each Gaussian component. The re-estimation formulas are consistent with our intuition. These MLE formulas calculate the weighted contribution of each data sample according to the mixture posterior probability γ_k^i.

In fact, VQ is an approximate version of EM algorithms. A traditional VQ with the Mahalanobis distance measure is equivalent to a mixture Gaussian VQ with the following conditions

$$c_k = 1/K \tag{4.109}$$

$$\gamma_k^i = \begin{cases} 1, & \mathbf{y}_i \in C_k \\ 0, & \text{otherwise} \end{cases} \tag{4.110}$$

The difference between VQ and the EM algorithm is that VQ performs a hard assignment of the data sample $\mathbf{y}_i$ to clusters (cells) while the EM algorithm performs a soft assignment of the data sample $\mathbf{y}_i$ to clusters. As discussed in Chapter 8, this difference carries over to the case of the Viterbi algorithm vs. the Baum-Welch algorithm in hidden Markov models.

4.5. CLASSIFICATION AND REGRESSION TREES

Classification and regression trees (CART) [15, 82] have been used in a variety of pattern recognition applications. Binary decision trees, with splitting questions attached to each node, provide an easy representation that interprets and predicts the structures of a set of data. The application of binary decision trees is much like playing the *number-guessing* game, where the examinee tries to deduce the chosen number by asking a series of binary number-comparing questions.

Consider a simple binary decision tree for height classification. Every person's data in the study may consist of several measurements, including race, gender, weight, age, occupation, and so on. The goal of the study is to develop a classification method to assign a person one of the following five height classes: *tall* (T), *medium-tall* (t), *medium* (M), *medium-shor t*(s) and *short* (S). Figure 4.14 shows an example of such a binary tree structure. With this binary decision tree, one can easily predict the height class for any new person (with all the measured data, but no height information) by traversing the binary trees. Traversing the binary tree is done through answering a series of yes/no questions in the traversed nodes with the measured data. When the answer is *yes*, the right branch is traversed next; otherwise, the left branch will be traversed instead. When the path ends at a leaf node, you can use its attached label as the height class for the new person. If you have the average height for each leaf node (computed by averaging the heights from those people who fall in the same leaf node during training), you can actually use the average height in the leaf node to predict the height for the new person.

This classification process is similar to a rule-based system where the classification is carried out by a sequence of decision rules. The choice and order of rules applied in a rule-based system is typically designed subjectively by hand through an introspective analysis based on the impressions and intuitions of a limited number of data samples. CART, on the other hand, provides an automatic and data-driven framework to construct the decision process based on objective criteria. Most statistical pattern recognition techniques are designed for data samples having a standard structure with homogeneous variables. CART is designed instead to handle data samples with high dimensionality, mixed data types, and nonstandard data structure. It has the following advantages over other pattern recognition techniques:

- CART can be applied to any data structure through appropriate formulation of the set of potential questions.
- The binary tree structure allows for compact storage, efficient classification, and easily understood interpretation of the predictive structure of the data.
- It often provides, without additional effort, not only classification and recognition, but also an estimate of the misclassification rate for each class.

- It not only handles missing data, but also is very robust to outliers and mislabeled data samples.

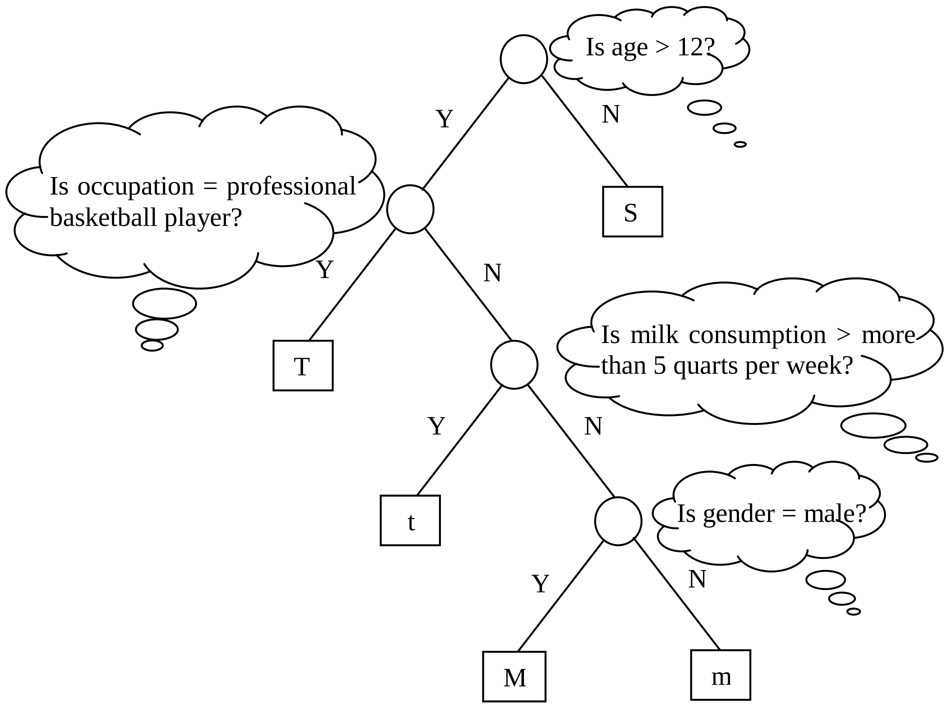
Is age > 12?
Y
N
S
Is occupation = professional basketball player?
Y
N
T
Is milk consumption > more than 5 quarts per week?
Y
N
t
Is gender = male?
Y
N
M
m

Figure 4.14 A binary tree structure for height classification

To construct a CART from the training samples with their classes (let's denote the set as $\Im$), we first need to find a set of questions regarding the measured variables; e.g., "*Is age > 12*?", "*Is occupation = professional basketball player*?", "*Is gender = male*?" and so on. Once the question set is determined, CART uses a greedy algorithm to generate the decision trees. All training samples $\Im$ are placed in the root of the initial tree. The *best question* is then chosen from the question set to split the root into two nodes. Of course, we need a measurement of how well each question splits the data samples to pick the best question. The algorithm recursively splits the most promising node with the best question until the right-sized tree is obtained. We describe next how to construct the question set, how to measure each split, how to grow the tree, and how to choose the right-sized tree.

4.5.1. Choice of Question Set

Assume that the training data has the following format:

$$\mathbf{x} = (x_1, x_2, \ldots x_N) \tag{4.111}$$

where each variable x_i is a discrete or continuous data type. We can construct a *standard set* of questions Q as follows:

1. Each question is about the value of only a single variable. Questions of this type are called *simple* or *singleton* questions.
2. If x_i is a discrete variable from the set $\{c_1, c_2, \ldots, c_K\}$, Q includes all questions of the following form:

$$\{\text{Is } x_i \in S?\} \tag{4.112}$$

where S is any subset of $\{c_1, c_2, \ldots, c_K,\}$

3. If x_i is a continuous variable, Q includes all questions of the following form:

$$\{\text{Is } x_i \le c?\} \text{ for } c \in (-\infty, \infty) \tag{4.113}$$

The question subset generated from discrete variables (in condition 2 above) is clearly a finite set ($2^{K-1}-1$). On the other hand, the question subset generated from continuous variables (in condition 3 above) seems to be an infinite set based on the definition. Fortunately, since the training data samples are finite, there are only finite number of distinct splits for the training data. For a continuous variable x_i, the data points in $\Im$ contain at most *M* distinct values $v_1, v_2, \ldots, v_M$. There are only at most *M* different splits generated by the set of questions in the form:

$$\{\text{Is } x_i \le c_n\} \qquad n = 1, 2, \ldots, M \tag{4.114}$$

where $c_n = \dfrac{v_{n-1} + v_n}{2}$ and $v_0 = 0$. Therefore, questions related to a continuous variable also form a finite subset. The fact that Q is a finite set allows the enumerating of all possible questions in each node during tree growing.

The construction of a question set is similar to that of rules in a rule-based system. Instead of using the all-possible question set Q, some people use knowledge selectively to pick a subset of Q, which is sensitive to pattern classification. For example, the vowel subset and consonant subset are a natural choice for these sensitive questions for phoneme classification. However, the beauty of CART is the ability to use all possible questions related to the measured variables, because CART has a statistical data-driven framework to determine the decision process (as described in subsequent Sections). Instead of setting some constraints on the questions (splits), most CART systems use all the possible questions for Q.

4.5.2. Splitting Criteria

A question in CART framework represents a split (partition) of data samples. All the leaf nodes (L in total) represent L disjoint subsets $A_1, A_2, \ldots, A_L$. Now we have the entire potential question set Q, the task is how to find the best question for a node split. The selection of the best question is equivalent to finding the best split for the data samples of the node.

Since each node t in the tree contains some training samples, we can compute the corresponding class probability density function $P(\omega \mid t)$. The classification process for the node can then be interpreted as a random process based on $P(\omega \mid t)$. Since our goal is classification, the objective of a decision tree is to reduce the uncertainty of the event being decided upon. We want the leaf nodes to be as pure as possible in terms of the class distribution. Let Y be the random variable of classification decision for data sample $\mathbf{X}$. We could define the weighted entropy for any node t as follows:

$$\bar{H}_t(Y) = H_t(Y)P(t) \tag{4.115}$$

$$H_t(Y) = -\sum_i P(\omega_i \mid t)\log P(\omega_i \mid t) \tag{4.116}$$

where $P(\omega_i \mid t)$ is the percentage of data samples for class *i* in node *t*; and $P(t)$ is the prior probability of visiting node *t* (equivalent to the ratio of number of data samples in node *t* and the total number of training data samples). With this weighted entropy definition, the splitting criterion is equivalent to finding the question which gives the greatest entropy reduction, where the entropy reduction for a question *q* to split a node *t* into leaves *l* and *r* can be defined as:

$$\Delta\bar{H}_t(q) = \bar{H}_t(Y) - (\bar{H}_l(Y) + \bar{H}_r(Y)) = \bar{H}_t(Y) - \bar{H}_t(Y \mid q) \tag{4.117}$$

The reduction in entropy is also the mutual information between Y and question q. The task becomes that of evaluating the entropy reduction $\Delta\bar{H}_q$ for each potential question (split), and picking the question with the greatest entropy reduction, that is,

$$q^* = \underset{q}{\operatorname{argmax}}\,(\Delta\bar{H}_t(q)) \tag{4.118}$$

If we define the entropy for a tree, T, as the sum of weighted entropies for all the terminal nodes, we have:

$$\bar{H}(T) = \sum_{t \text{ is terminal}} \bar{H}_t(Y) \tag{4.119}$$

It can be shown that the tree-growing (splitting) process repeatedly reduces the entropy of the tree. The resulting tree thus has a better classification power. For continuous pdf, likelihood gain is often used instead, since there is no straightforward entropy measurement [43]. Suppose one specific split divides the data into two groups, $\mathbf{X}_1$ and $\mathbf{X}_2$, which can then be used to train two Gaussian distributions $N(\boldsymbol{\mu}_1, \Sigma_1)$ and $N(\boldsymbol{\mu}_2, \Sigma_2)$. The log-likelihoods for generating these two data groups are:

$$L_1(\mathbf{X}_1 \mid N) = \log \prod_{\mathbf{x}_1} N(\mathbf{x}_1, \boldsymbol{\mu}_1, \Sigma_1) = (d \log 2\pi + \log|\Sigma_1| + d)a/2 \tag{4.120}$$

$$L_2(\mathbf{X}_2 \mid N) = \log \prod_{\mathbf{x}_2} N(\mathbf{x}_2, \boldsymbol{\mu}_2, \Sigma_2) = (d \log 2\pi + \log|\Sigma_1| + d)b/2 \tag{4.121}$$

where *d* is the dimensionality of the data; and *a* and *b* are the sample counts for the data groups $\mathbf{X}_1$ and $\mathbf{X}_2$ respectively. Now if the entire data $\mathbf{X}_1$ and $\mathbf{X}_2$ are merged into one group and modeled by one Gaussian $N(\boldsymbol{\mu}, \Sigma)$, according to MLE, we have

$$\boldsymbol{\mu} = \frac{a}{a+b}\boldsymbol{\mu}_1 + \frac{b}{a+b}\boldsymbol{\mu}_2 \tag{4.122}$$

$$\Sigma = \frac{a}{a+b}\left[\Sigma_1 + (\boldsymbol{\mu}_1 - \boldsymbol{\mu})(\boldsymbol{\mu}_1 - \boldsymbol{\mu})^t\right]\Sigma_1 + \frac{b}{a+b}\left[\Sigma_2 + (\boldsymbol{\mu}_2 - \boldsymbol{\mu})(\boldsymbol{\mu}_2 - \boldsymbol{\mu})^t\right] \tag{4.123}$$

Thus, the likelihood gain of splitting the data **X** into two groups $\mathbf{X}_1$ and $\mathbf{X}_2$ is:

$$\begin{aligned} \Delta \overline{L}_t(q) &= L_1(\mathbf{X}_1 \mid N) + L_2(\mathbf{X}_2 \mid N) - L_{\mathbf{x}}(\mathbf{X} \mid N) \\ &= (a+b)\log|\Sigma| - a\log|\Sigma_1| - b\log|\Sigma_2| \end{aligned} \tag{4.124}$$

For regression purposes, the most popular splitting criterion is the mean squared error measure, which is consistent with the common *least squared* regression methods. For instance, suppose we need to investigate the real height as a regression function of the measured variables in the height study. Instead of finding height classification, we could simply use the average height in each node to predict the height for any data sample. Suppose Y is the actual height for training data $\mathbf{X}$, then overall regression (prediction) error for a node *t* can be defined as:

$$E(t) = \sum_{\mathbf{X} \in t} |Y - d(\mathbf{X})|^2 \tag{4.125}$$

where $d(\mathbf{X})$ is the regression (predictive) value of Y

Now, instead of finding the question with greatest entropy reduction, we want to find the question with largest squared error reduction. That is, we want to pick the question q^* that maximizes:

$$\Delta E_t(q) = E(t) - (E(l) + E(r)) \tag{4.126}$$

where *l* and *r* are the leaves of node *t*. We define the expected square error $V(t)$ for a node *t* as the overall regression error divided by the total number of data samples in the node.

$$V(t) = E\left(\sum_{\mathbf{X}\in t} |Y - d(\mathbf{X})|^2 \right) = \frac{1}{N(t)} \sum_{\mathbf{X}\in t} |Y - d(\mathbf{X})|^2 \tag{4.127}$$

Note that $V(t)$ is actually the variance estimate of the height, if $d(\mathbf{X})$ is made to be the average height of data samples in the node. With $V(t)$, we define the weighted squared error $\bar{V}(t)$ for a node *t* as follows.

$$\bar{V}(t) = V(t)P(t) = \left(\frac{1}{N(t)} \sum_{\mathbf{X}\in t} |Y - d(\mathbf{X})|^2 \right) P(t) \tag{4.128}$$

Finally, the splitting criterion can be rewritten as:

$$\Delta \bar{V}_t(q) = \bar{V}(t) - (\bar{V}(l) + \bar{V}(r)) \tag{4.129}$$

Based on Eqs. (4.117) and (4.129), one can see the analogy between entropy and variance in the splitting criteria for CART. The use of entropy or variance as splitting criteria is under the assumption of uniform misclassification costs and uniform prior distributions. When nonuniform misclassification costs and prior distributions are used, some other splitting might be used for splitting criteria. Noteworthy ones are *Gini index of diverity* and *twoing rule*. Those interested in alternative splitting criteria can refer to [11, 15].

For a wide range of splitting criteria, the properties of the resulting CARTs are empirically insensitive to these choices. Instead, the criterion used to get the right-sized tree is much more important. We discuss this issue in Section 4.5.6.

4.5.3. Growing the Tree

Given the question set Q and splitting criteria $\Delta \bar{H}_t(q)$, the tree-growing algorithm starts from the initial root-only tree. At each node of tree, the algorithm searches through the variables one by one, from x_1 to x_N. For each variable, it uses the splitting criteria to find the best question (split). Then it can pick the best question out of the *N* best single-variable questions. The procedure can continue splitting each node until either of the following conditions is met for a node:

1. No more splits are possible; that is, all the data samples in the node belong to the same class;
2. The greatest entropy reduction of best question (split) fall below a pre-set threshold β, i.e.:

$$\max_{q\in Q} \Delta \bar{H}_t(q) < \beta \tag{4.130}$$

3. The number of data samples falling in the leaf node *t* is below some threshold α. This is to assure that there are enough training tokens for each leaf node if one needs to estimate some parameters associated with the node.

When a node cannot be further split, it is declared a terminal node. When all active (non-split) nodes are terminal, the tree-growing algorithm stops.

The algorithm is greedy because the question selected for any given node is the one that seems to be the best, without regard to subsequent splits and nodes. Thus, the algorithm constructs a tree that is locally optimal, but not necessarily globally optimal (but hopefully globally *good enough*). This tree-growing algorithm has been successfully applied in many applications [5, 39, 60]. A dynamic programming algorithm for determining global optimality is described in [78]; however, it is suitable only in restricted applications with relatively few variables.

4.5.4. Missing Values and Conflict Resolution

Sometimes, the available data sample $\mathbf{X} = (x_1, x_2, \ldots x_N)$ has some value x_j missing. This missing-value case can be handled by the use of *surrogate questions* (*splits*). The idea is intuitive. We define a similarity measurement between any two questions (splits) q and $\tilde{q}$ of a node t. If the best question of node t is the question q on the variable x_i, find the question $\tilde{q}$ that is most similar to q on a variable other than x_i. $\tilde{q}$ is our best surrogate question. Similarly, we find the second-best surrogate question, third-best and so on. The surrogate questions are considered as the backup questions in the case of missing x_i values in the data samples. The surrogate question is used in the descending order to continue tree traversing for those data samples. The surrogate question gives CART unique ability to handle the case of missing data. The similarity measurement is basically a measurement reflecting the similarity of the class probability density function [15].

When choosing the best question for splitting a node, several questions on the same variable x_i may achieve the same entropy reduction and generate the same partition. As in rule-based problem solving systems, a *conflict resolution* procedure [99] is needed to decide which question to use. For example, discrete questions q_1 and q_2 have the following format:

$$q_1 \ : \ \{\text{Is } x_i \in S_1 \,?\} \tag{4.131}$$

$$q_2 \ : \ \{\text{Is } x_i \in S_2 \,?\} \tag{4.132}$$

Suppose S_1 is a subset of S_2, and one particular node contains only data samples whose x_i value contains only values in S_1, but no other. Now question q_1 or q_2 performs the same splitting pattern and therefore achieves exactly the same amount of entropy reduction. In this case, we call q_1 a sub-question of question q_2, because q_1 is a more specific version.

A *specificity ordering* conflict resolution strategy is used to favor the discrete question with fewer elements because it is more specific to the current node. In other words, if the elements of a question are a subset of the elements of another question with the same entropy reduction, the question with the subset of elements is preferred. Preferring more specific questions will prevent decision trees from over-generalizing. The specificity ordering conflict resolution can be implemented easily by presorting the set of discrete questions by the number of elements they contain in descending order, before applying them to decision trees. A similar specificity ordering conflict resolution can also be implemented for continuous-variable questions.

4.5.5. Complex Questions

One problem with allowing only simple questions is that the data may be over-fragmented, resulting in similar leaves in different locations of the tree. For example, when the best question (rule) to split a node is actually a composite question of the form "*Is* $x_i \in S_1$?" or "*Is* $x_i \in S_2$?", a system allowing only simple questions will generate two separate questions to split the data into three clusters rather than two as shown in Figure 4.15. Also data for which the answer is *yes* are inevitably fragmented across two shaded nodes. This is inefficient and ineffective since these two very similar data clusters may now both contain insufficient training examples, which could potentially handicap the future tree growing. Splitting data unnecessarily across different nodes leads to unnecessary computation, redundant clusters, reduced trainability, and less accurate entropy reduction.

We deal with this problem by using a composite-question construction [38, 40]. It involves conjunctive and disjunctive combinations of all questions (and their negations). A composite question is formed by first growing a tree with simple questions only and then clustering the leaves into two sets. Figure 4.16 shows the formation of one composite question. After merging, the structure is still a binary question. To construct the composite question, multiple OR operators are used to describe the composite condition leading to either one of the final clusters, and AND operators are used to describe the relation within a particular route. Finally, a Boolean reduction algorithm is used to simplify the Boolean expression of the composite question.

To speed up the process of constructing composite questions, we constrain the number of leaves or the depth of the binary tree through heuristics. The most frequently used heuristics is the limitation of the depth when searching a composite question. Since composite questions are essentially binary questions, we use the same greedy tree-growing algorithm to find the best composite question for each node and keep growing the tree until the stop criterion is met. The use of composite questions not only enables flexible clustering, but also improves entropy reduction. Growing the sub-tree a little deeper before constructing the composite question may achieve longer-range optimum, which is preferable to the local optimum achieved in the original greedy algorithm that used simple questions only.

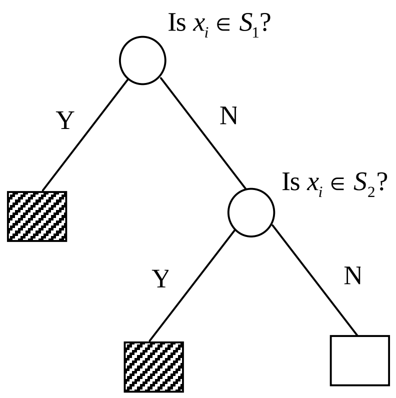

Figure 4.15 A over-split tree for question “*Is* $x_i \in S_1$?” or “*Is* $x_i \in S_2$?”

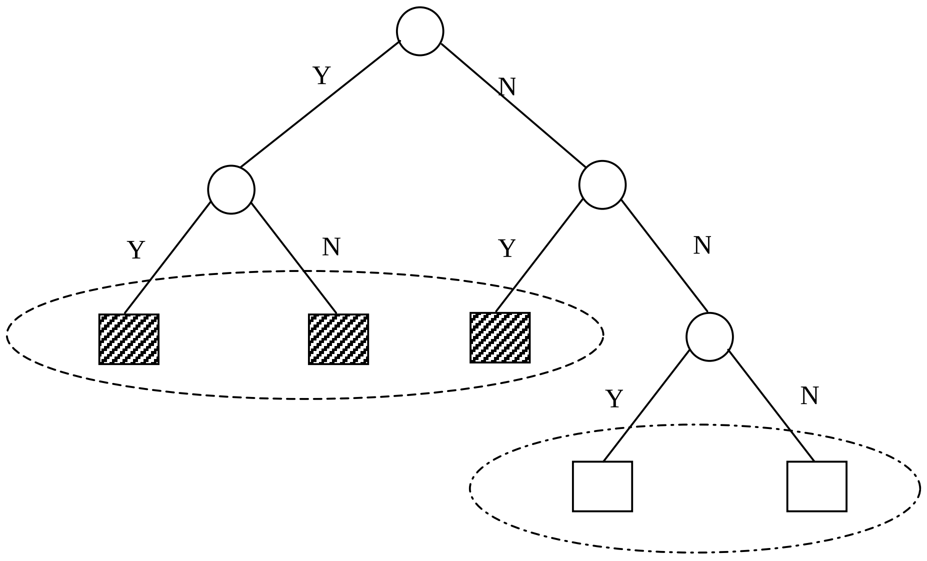

Figure 4.16 The formation of a composite question from simple questions

The construction of composite questions can also be applied to continuous variables to obtained complex rectangular partitions. However, some other techniques are used to obtain general partitions generated by hyperplanes not perpendicular to the coordinate axes. Questions typically have a linear combination of continuous variables in the following form [15]:

$$\{\text{Is } \sum_i a_i x_i \le c\,?\} \tag{4.133}$$

4.5.6. The Right-Sized Tree

One of the most critical problems for CART is that the tree may be strictly tailored to the training data that has no generalization capability. When you split a leaf node in the tree to get entropy reduction until each leaf node contains data from one single class, that tree possesses a zero percent classification error on the training set. This is an over-optimistic estimate of the test-set misclassification rate. Independent test sample estimation or cross-validation is often used to prevent decision trees from over-modeling idiosyncrasies of the training data. To get a right-sized tree, you can minimize the misclassification rate for future independent test data.

Before we describe the solution for finding the right sized tree, let's define a couple of useful terms. Naturally we will use the plurality rule $\delta(t)$ to choose the class for a node *t*:

$$\delta(t) = \underset{i}{\operatorname{argmax}}\, P(\omega_i \mid t) \tag{4.134}$$

Similar to the notation used in Bayes' decision theory, we can define the misclassification rate $R(t)$ for a node *t* as:

$$R(t) = r(t)P(t) \tag{4.135}$$

where $r(t) = 1 - \max_i P(\omega_i \mid t)$ and $P(t)$ is the frequency (probability) of the data falling in node *t*. The overall misclassification rate for the whole tree *T* is defined as:

$$R(T) = \sum_{t \in \tilde{T}} R(t) \tag{4.136}$$

where $\tilde{T}$ represents the set of terminal nodes. If a nonuniform misclassification cost $c(i \mid j)$, the cost of misclassifying class *j* data as class *i* data, is used, $r(t)$ is redefined as:

$$r(t) = \min_i \sum_j c(i \mid j) P(j \mid t) \tag{4.137}$$

As we mentioned, $R(T)$ can be made arbitrarily small (eventually reduced to zero) for the training data if we keep growing the tree. The key now is how we choose the tree that can minimize $R^*(T)$, which is denoted as the misclassification rate of independent test data. Almost no tree initially grown can perform well on independent test data. In fact, using more complicated stopping rules to limit the tree growing seldom works, and it is either stopped too soon at some terminal nodes, or continued too far in other parts of the tree. Instead of inventing some clever stopping criteria to stop the tree growing at the right size, we let the tree over-grow (based on rules in Section 4.5.3). We use a pruning strategy to gradually cut back the tree until the minimum $R^*(T)$ is achieved. In the next section we describe an algorithm to prune an over-grown tree, *minimum cost-complexity pruning*.

4.5.6.1. Minimum Cost-Complexity Pruning

To prune a tree, we need to find a subtree (or branch) that makes the least impact in terms of a cost measure, whether it is pruned or not. This candidate to be pruned is called the *weakest subtree*. To define such a weakest subtree, we first need to define the cost measure.

DEFINITION 1: For any sub-tree T of $T_{\max}$ ($T \prec T_{\max}$), let $|\tilde{T}|$ denote the number of terminal nodes in tree T.

DEFINITION 2: Let $\alpha \ge 0$ be a real number called the *complexity parameter*. The cost-complexity measure can be defined as:

$$R_\alpha(T) = R(T) + \alpha |\tilde{T}| \tag{4.138}$$

DEFINITION 3: For each α, define the minimal cost-complexity subtree $T(\alpha) \prec T_{\max}$ that minimizes $R_\alpha(T)$, that is,

$$T(\alpha) = \arg\min_{T \prec T_{\max}} R_\alpha(T) \tag{4.139}$$

Based on DEFINITION 3**,** if α is small, the penalty for having a large tree is small and $T(\alpha)$ will be large**.** In fact, $T(0)$ = $T_{\max}$ because $T_{\max}$ has a zero misclassification rate, so it will minimize $R_o(T)$. On the other hand, when α increases**,** $T(\alpha)$ becomes smaller and smaller. For a sufficient large α**,** $T(\alpha)$ may collapse into a tree with only the root. The increase of α produces a sequence of pruned trees and it is the basis of the pruning process. The pruning algorithm rests on two theorems. The first is given as follows.

THEOREM 1: For every value of α**,** there exists a unique minimal cost-complexity subtree $T(\alpha)$ as defined in Definition 3.[12]

To progressively prune the tree, we need to find the weakest subtree (node). The idea of a weakest subtree is the following: *if we collapse the weakest subtree into a single terminal node, the cost-complexity measure would increase least.* For any node t in the tree T, let $\{t\}$ denote the subtree containing only the node t, and T_t denote the branch starting at node t. Then we have

$$R_\alpha(T_t) = R(T_t) + \alpha |\tilde{T}_t| \tag{4.140}$$

$$R_\alpha(\{t\}) = R(t) + \alpha \tag{4.141}$$

When α is small, T_t has a smaller cost-complexity than the single-node tree $\{t\}$. However, when α increases to a point where the cost-complexity measures for T_t and $\{t\}$

[12] You can find the proof to this in [15].

are the same, it makes sense to collapse T_t into a single terminal node $\{t\}$. Therefore, we decide the critical value for α by solving the following inequality:

$$R_\alpha(T_t) \le R_\alpha(\{t\}) \tag{4.142}$$

We obtain:

$$\alpha \le \frac{R(t) - R(T_t)}{|\tilde{T}_t| - 1} \tag{4.143}$$

Based on Eq. (4.143), we define a measurement $\eta(t)$ for each node *t* in tree *T*:

$$\eta(t) = \begin{cases} \dfrac{R(t) - R(T_t)}{|\tilde{T}_t| - 1}, & t \notin \tilde{T} \\ +\infty, & t \in \tilde{T} \end{cases} \tag{4.144}$$

Based on measurement $\eta(t)$, we then define the weakest subtree T_{t_1} as the tree branch starting at the node t_1 such that

$$t_1 = \arg\min_{t \in T} \ \eta(t) \tag{4.145}$$

$$\alpha_1 = \eta(t_1) \tag{4.146}$$

As α increases, the node t_1 is the first node such that $R_\alpha(\{t\})$ becomes equal to $R_\alpha(T_t)$. At this point, it would make sense to prune subtree T_{t_1} (collapse T_{t_1} into a single-node subtree $\{t_1\}$), and α_1 is the value of α where the pruning occurs.

Now the tree T after pruning is referred to as T_1, i.e.,

$$T_1 = T - T_{t_1} \tag{4.147}$$

We then use the same process to find the weakest subtree T_{t_2} in T_1 and the new pruning point α_2. After pruning away T_{t_2} from T_1 to form the new pruned tree T_2, we repeat the same process to find the next weakest subtree and pruning point. If we continue the process, we get a sequence of decreasing pruned trees:

$$T \succ T_1 \succ T_2 \succ T_2 \cdots \succ \{r\} \tag{4.148}$$

where $\{r\}$ is the single-node tree containing the root of tree T with corresponding pruning points:

$$\alpha_0 < \alpha_1 < \alpha_2 < \alpha_3 < \cdots \tag{4.149}$$

where $\alpha_0 = 0$

With the process above, the following theorem (which is basic for the minimum cost-complexity pruning) can be proved.

THEOREM 2 : Let T_0 be the original tree T .

$$\text{For } k \geq 0\,,\ \alpha_k \leq \alpha < \alpha_{k+1}\,,\ T(\alpha) = T(\alpha_k) = T_k \tag{4.150}$$

4.5.6.2. Independent Test Sample Estimation

The minimum cost-complexity pruning algorithm can progressively prune the over-grown tree to form a decreasing sequence of subtrees $T \succ T_1 \succ T_2 \succ T_2 \cdots \succ \{r\}$, where $T_k = T(\alpha_k)$, $\alpha_0 = 0$ and $T_0 = T$. The task now is simply to choose one of those subtrees as the optimal-sized tree. Our goal is to find the optimal-sized tree that minimizes the misclassification for independent test set $R^*(T)$. When the training set $\Im$ is abundant, we can afford to set aside an independent test set $\Re$ from the training set. Usually $\Re$ is selected as one third of the training set $\Im$. We use the remaining two thirds of the training set $\Im - \Re$ (still abundant) to train the initial tree T and apply the minimum cost-complexity pruning algorithm to attain the decreasing sequence of subtrees $T \succ T_1 \succ T_2 \succ T_2 \cdots \succ \{r\}$. Next, the test set $\Re$ is run through the sequence of subtrees to get corresponding estimates of test-set misclassification $R^*(T), R^*(T_1), R^*(T_2), \cdots, R^*(\{r\})$. The optimal-sized tree T_{k^*} is then picked as the one with minimum test-set misclassification measure, i.e.:

$$k^* = \arg\min_k R^*(T_k) \tag{4.151}$$

The *independent test sample estimation* approach has the drawback that it reduces the effective training sample size. This is why it is used only when there is abundant training data. Under most circumstances where training data is limited, *cross-validation* is often used.

4.5.6.3. Cross-Validation

CART can be pruned via *v-fold cross-validation*. It follows the same principle of cross validation described in Section 4.2.3. First it randomly divides the training set $\Im$ into *v* disjoint subset $\Im_1, \Im_2, \cdots, \Im_v$, each containing roughly the same data samples. It then defines the *i*th training set

$$\Im^i = \Im - \Im_i \qquad i = 1, 2, \ldots, v \tag{4.152}$$

so that $\Im^i$ contains the fraction $(v-1)/v$ of the original training set. v is usually chosen to be large, like 10.

In *v*-fold cross-validation, *v* auxiliary trees are grown together with the main tree T grown on $\Im$. The *i*th tree is grown on the *i*th training set $\Im^i$. By applying minimum cost-complexity pruning, for any given value of the cost-complexity parameter α, we can obtain the corresponding minimum cost-complexity subtrees $T(\alpha)$ and $T^i(\alpha)$, $i = 1, 2, \ldots, v$. According to Theorem 2 in Section 4.5.6.1, those minimum cost-complexity subtrees will form $v+1$ sequences of subtrees:

$$T \succ T_1 \succ T_2 \succ T_2 \cdots \succ \{r\} \text{ and} \tag{4.153}$$

$$T^i \succ T_1^i \succ T_2^i \succ T_3^i \cdots \succ \{r^i\} \qquad i = 1, 2, \ldots, v \tag{4.154}$$

ALGORITHM 4.5 *THE CART ALGORITHM*

Step 1: Question Set: Create a standard set of questions Q that consists of all possible questions about the measure variables.
Step 2: Splitting Criterion: Pick a splitting criterion that can evaluate all the possible questions in any node. Usually it is either entropy-like measurement for classification trees or mean square errors for regression trees.
Step 3: Initialization: Create a tree with one (root) node, consisting of all training samples.
Step 4: Split Candidates: Find the best composite question for each terminal node:

a. Generate a tree with several simple-question splits as described in Section 4.5.3.
b. Cluster leaf nodes into two classes according to the same splitting criterion.
c. Based on the clustering done in (b), construct a corresponding composite question.

Step 5: Split**:** Out of all the split candidates in Step 4, split the one with best criterion.
Step 6: Stop Criterion**:** If all the leaf node containing data samples from the same class or all the potential splits generate improvement fall below a pre-set threshold β, go to Step 7; otherwise go to Step 4.
Step 7: Use *independent test sample estimate* or *cross-validation estimate* to prune the original tree into the optimal size.

The basic assumption of cross-validation is that the procedure is *stable* if v is large. That is, $T(\alpha)$ should have the same classification accuracy as $T^i(\alpha)$. Although we cannot directly estimate the test-set misclassification for the main tree $R^*(T(\alpha))$, we could approximate it via the test-set misclassification measure $R^*(T^i(\alpha))$, since each data sample in $\Im$ occurs in one and only one test set $\Im_i$. The *v*-fold cross-validation estimate $R^{CV}(T(\alpha))$ can be computed as:

$$R^{CV}(T(\alpha)) = \frac{1}{v}\sum_{i=1}^{v} R^*(T^i(\alpha)) \tag{4.155}$$

Similar to Eq. (4.151), once $R^{CV}(T(\alpha))$ is computed, the optimal *v*-fold cross-validation tree $T_{k^{CV}}$ can be found through

$$k^{CV} = \arg\min_k R^{CV}(T_k) \tag{4.156}$$

Cross-validation is computationally expensive in comparison with independent test sample estimation, though it makes more effective use of all training data and reveals useful information regarding the stability of the tree structure. Since the auxiliary trees are grown on a smaller training set (a fraction $v-1/v$ of the original training data), they tend to have a higher misclassification rate. Therefore, the cross-validation estimates $R^{CV}(T)$ tend to be an over-estimation of the misclassification rate. The algorithm for generating a CART tree is illustrated in Algorithm 4.5.

4.6. HISTORICAL PERSPECTIVE AND FURTHER READING

Pattern recognition is a multidisciplinary field that comprises a broad body of loosely related knowledge and techniques. Historically, there are two major approaches to pattern recognition – the statistical and the syntactical approaches. Although this chapter is focused on the statistical approach, syntactical pattern recognition techniques, which aim to address the limitations of the statistical approach in handling contextual or structural information, can be complementary to statistical approaches for spoken language processing, such as parsing. Syntactic pattern recognition is based on the analogy that complex patterns can be decomposed recursively into simpler subpatterns, much as a sentence can be decomposed into words and letters. Fu [24] provides an excellent book on syntactic pattern recognition.

The foundation of statistical pattern recognition is Bayesian theory, which can be traced back to the 18th century [9, 54] and its invention by the British mathematician Thomas Bayes (1702-1761). Chow [20] was the first to use Bayesian decision theory for pattern recognition. Statistical pattern recognition has been used successfully in a wide range of applications, from optical/handwritten recognition [13, 96], to speech recognition [7, 86] and to medical/machinery diagnosis [1, 27]. The books by Duda et al. [22] and Fukunaga [25] are two classic treatments of statistical pattern recognition. Duda et al. have a second edition of the classic pattern recognition book [23] that includes many up-to-date topics.

MLE and MAP are two most frequently used estimation methods for pattern recognition because of their simplicity and efficiency. In Chapters 8 and 9, they play an essential role in model parameter estimation. Estimating the recognition performance and comparing different recognition systems are important subjects in pattern recognition. The importance of a large number of test samples was reported in [49]. McNemar's test is dated back to the 1940s [66]. The modification of the test for continuous speech recognition systems presented in this chapter is based on an interesting paper [30] that contains a general discussion on using hypothesis-testing methods for continuous speech recognition.

Gradient descent is fundamental for most discriminant estimation methods, including MMIE, MCE, and neural networks. The history of gradient descent can be traced back to

Newton's method for root finding [72, 81]. Both the book by Duda et al. [23] and the paper by Juang et al. [48] provide a good description of gradient descent. MMIE was first proposed in [16, 71] for the speech recognition problem. According to these two works, MMIE is more robust than MLE to incorrect model assumptions. MCE was first formulated by Juang et al. [48] and successfully applied to small-vocabulary speech recognition [47].

The modern era of neural networks was brought to the scientific community by McCulloch and Pitts. In the pioneering paper [64], McCulloch and Pitts laid out the mathematical treatment of the behavior of networks of simple neurons. The most important result they showed is that a network would compute any computable function. John von Neumann was influenced by this paper to use switch-delay elements derived from the McCulloch-Pitts neuron in the construction of the EDVAC (Electronic Discrete Variable Automatic Computer) that was developed based on ENIAC (Electronic Numerical Integrator and Computer) [2, 35]. The ENIAC was the famous first general-purpose electronic computer built at the Moore School of Electrical Engineering at the University of Pennsylvania from 1943 to 1946 [31]. The two-layer perceptron work [87] by Rosenblatt, was the first to provide rigorous proofs about perceptron convergence. A 1969 book by Minsky and Papert [68] reveals that there are fundamental limits to what single-layer perceptrons can compute. It was not until the 1980s that the discovery of multi-layer perceptrons (with hidden layers and nonlinear threshold functions) and back-propagation [88] reawakened interest in neural networks. The two-volume PDP book [90, 91], *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, edited by Rummelhart and McClelland, brought the back-propagation learning method to the attention of the widest audience. Since then, various applications of neural networks in diverse domains have been developed, including speech recognition [14, 58], speech production and perception [93, 94], optical and handwriting character recognition [55, 92], visual recognition [26], game playing [97], and natural language processing [63]. There are several good textbooks for neural networks. In particular, the book by Haykin [35] provides a very comprehensive coverage of all foundations of neural networks. Bishop [12] provides a thoughtful treatment of neural networks from the perspective of pattern recognition. Short, concise tutorial papers on neural networks can be found in [44, 57].

Vector quantization originated from speech coding [17, 32, 45, 61]. The *k*-means algorithm was introduced by Lloyd [59]. Over the years, there have been many variations of VQ, including fuzzy VQ [10], learning VQ (LVQ) [51], and supervised VQ [18, 42]. The first published investigation toward the EM-like algorithm for incomplete data learning can be attributed to Pearson [79]. The modern EM algorithm is formalized by Dempster, Laird, and Rubin [21]. McLachlan and Krishnan [65] provide a thorough overview and history of the EM algorithm. The convergence of the EM algorithm is an interesting research topic and Wu [100] has an extensive description of the rate of convergence. The EM algorithm is the basis for all unsupervised learning that includes hidden variables. The famous HMM training algorithm, as described in Chapter 8, is based on the EM algorithm.

CART uses a very intuitive and natural principle of sequential questions and answers, which can be traced back to 1960s [70]. The popularity of CART is attributed to the book by Breiman *et al.* [15]. Quinlan proposed some interesting variants of CART, like ID3 [82] and C4.5 [84]. CART has recently been one of the most popular techniques in machine learning.

Mitchell includes a good overview chapter on the latest CART techniques in his machine-learning book [69]. In addition to the strategies of node splitting and pruning mentioned in this chapter, [62] used a very interesting approach for splitting and pruning criteria based on a statistical significance testing of the data's distributions. Moreover, [28] proposed an iterative expansion pruning algorithm which is believed to perform as well as cross-validation pruning and yet is computationally cheaper [52]. CART has been successfully used in a variety of spoken language applications such as letter-to-sound conversion [46, 60], allophone model clustering [8, 38, 39], language models [5], automatic rule generation [83], duration modeling of phonemes [74, 80], and supervised vector quantization [67].

REFERENCES

[1] Albert, A. and E.K. Harris, *Multivariate Interpretation of Clinical Laboratory Data*, 1987, New York, Marcel Dekker.

[2] Aspray, W. and A. Burks, "Papers of John von Neumann on Computing and Computer Theory" in *Charles Babbage Institute Reprint Series for the History of Computing* 1986, Cambridge, MA, MIT Press.

[3] Atal, B.S. and M.R. Schroeder, "Predictive Coding of Speech Signals and Subjective Error Criteria," *IEEE Trans. on Acoustics, Speech and Signal Processing,* 1979, **ASSP-27**(3), pp. 247-254.

[4] Bahl, L.R., *et al.*, "A New Algorithm for the Estimation of Hidden Markov Model Parameters," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1988, New York, NY pp. 493-496.

[5] Bahl, L.R., *et al.*, "A Tree-Based Statistical Language Model for Natural Language Speech Recognition," *IEEE Trans. on Acoustics, Speech, and Signal Processing,* 1989, **37**(7), pp. 1001-1008.

[6] Bahl, L.R., *et al.*, "Estimating Hidden Markov Model Parameters so as to Maximize Speech Recognition Accuracy," *IEEE Trans. on Speech and Audio Processing*, 1993, **1**(1), pp. 77-83.

[7] Bahl, L.R., F. Jelinek, and R.L. Mercer, "A Maximum Likelihood Approach to Continuous Speech Recognition," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 1983, **5**(2), pp. 179-190.

[8] Bahl, L.R., *et al.*, "Decision Trees for Phonological Rules in Continuous Speech" in *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing* 1991, Toronto, Canada, pp. 185-188.

[9] Bayes, T., "An Essay Towards Solving a Problem in the Doctrine Of Chances," *Philosophical Tansactions of the Royal Society*, 1763, **53**, pp. 370-418.

[10] Bezdek, J., *Pattern Recognition with Fuzzy Objective Function Algorithms*, 1981, New York, NY, Plenum Press.

[11] Bhargava, T.N. and V.R.R. Uppuluri, "Sampling Distribution of Gini's Index of Diversity," *Applied Mathematics and Computation*, 1977, **3**, pp. 1-24.

[12] Bishop, C.M., *Neural Networks for Pattern Recognition*, 1995, Oxford, UK, Oxford University Press.

[13] Blesser, B., *et al.*, "A Theoretical Approach for Character Recognition Based on Phenomenological Attributes," *Int. Journal of Man-Machine Studies*, 1974, **6**(6), pp. 701-714.

[14] Bourlard, H. and N. Morgan, *Connectionist Speech Recognition - A Hybrid Approach*, 1994, Boston, MA, Kluwer Academic Publishers.

[15] Breiman, L., *et al.*, *Classification and Regression Trees*, 1984, Pacific Grove, CA, Wadsworth.

[16] Brown, P.F., *The Acoustic-Modeling Problem in Automatic Speech Recognition*, PhD Thesis in *Computer Science Department* 1987, Carnegie Mellon University, Pittsburgh, PA.

[17] Buzo, A., *et al.*, "Speech Coding Based upon Vector Quantization," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1980, **28**(5), pp. 562-574.

[18] Cerf, P.L., W. Ma, and D.V. Compernolle, "Multilayer Perceptrons as Labelers for Hidden Markov Models," *IEEE Trans. on Speech and Audio Processing*, 1994, **2**(1), pp. 185-193.

[19] Chang, P.C. and B.H. Juang, "Discriminative Training of Dynamic Programming Based Speech Recognizers," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1992, San Fancisco.

[20] Chow, C.K., "An Optimum Character Recognition System Using Decision Functions," *IRE Trans.*, 1957, pp. 247-254.

[21] Dempster, A.P., N.M. Laird, and D.B. Rubin, "Maximum-Likelihood from Incomplete Data via the EM Algorithm," *Journal of Royal Statistical Society ser. B*, 1977, **39**, pp. 1-38.

[22] Duda, R.O. and P.E. Hart, *Pattern Classification and Scene Analysis*, 1973, New York, N.Y., John Wiley and Sons.

[23] Duda, R.O., D.G. Stork, and P.E. Hart, *Pattern Classification and Scene Analysis : Pattern Classification*, 2nd ed, 1999, John Wiley & Sons.

[24] Fu, K.S., *Syntactic Pattern Recognition and Applications*, 1982, Englewood Cliffs, NJ, Prentice Hall.

[25] Fukunaga, K., *Introduction to Statistical Pattern Recognition*, Second ed, 1990, Orlando, FL, Academic Press.

[26] Fukushima, K., S. Miykake, and I. Takayuki, "Neocognition: A Neural Network Model for a Mechanism of Visual Pattern Recognition," *IEEE Trans. on Systems, Man and Cybernetics*, 1983, **SMC-13**(5), pp. 826-834.

[27] Gastwirth, J.L., "The Statistical Precision of Medical Screening Procedures: Application to Polygraph and AIDS Antibodies Test Data (with Discussion)," *Statistics Science*, 1987, **2**, pp. 213-238.

[28] Gelfand, S., C. Ravishankar, and E. Delp, "An Iterative Growing and Pruning Algorithm for Classification Tree Design," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 1991, **13**(6), pp. 163-174.

[29] Gersho, A., "On the Structure of Vector Quantization," *IEEE Trans. on Information Theory*, 1982, **IT-28**, pp. 256-261.

[30] Gillick, L. and S.J. Cox, "Some Statistical Issues in the Comparison of Speech Recognition Algorithms," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1989, Glasgow, Scotland, UK, IEEE pp. 532-535.

[31] Goldstine, H., *The Computer from Pascal to von Neumann*, 2nd ed, 1993, Princeton, NJ, Princeton University Press.

[32] Gray, R.M., "Vector Quantization," *IEEE ASSP Magazine*, 1984, **1**(April), pp. 4-29.

[33] Gray, R.M. and E.D. Karnin, "Multiple Local Optima in Vector Quantizers," *IEEE Trans. on Information Theory*, 1982, **IT-28**, pp. 256-261.

[34] Hartigan, J.A., *Clustering Algorithm*, 1975, New York, J. Wiley.

[35] Haykin, S., *Neural Networks: A Comprehensive Foundation*, 2dn ed, 1999, Upper Saddle River, NJ, Prentice-Hall.

[36] Hedelin, P., P. Knagenhjelm, and M. Skoglund, "Vector Quantization for Speech Transmission" in *Speech Coding and Synthesis*, W.B. Kleijn and K.K. Paliwal, eds. 1995, Amsterdam, pp. 311-396, Elsevier.

[37] Hinton, G.E., "Connectionist Learning Procedures," *Artificial Intelligence*, 1989, **40**, pp. 185--234.

[38] Hon, H.W., *Vocabulary-Independent Speech Recognition: The VOCIND System*, Ph.D Thesis in *Department of Computer Science* 1992, Carnegie Mellon University, Pittsburgh.

[39] Hon, H.W. and K.F. Lee, "On Vocabulary-Independent Speech Modeling," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1990, Albuquerque, NM pp. 725-728.

[40] Hon, H.W. and K.F. Lee, "Vocabulary-Independent Subword Modeling and Adaptation," *IEEE Workshop on Speech Recognition*, 1991, Arden House, Harriman, NY.

[41] Huang, X.D., Y. Ariki, and M.A. Jack, *Hidden Markov Models for Speech Recognition*, 1990, Edinburgh, U.K., Edinburgh University Press.

[42] Hunt, M.J., *et al.*, "An Investigation of PLP and IMELDA Acoustic Representations and of Their Potential for Combination" in *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing* 1991, Toronto, Canada, pp. 881-884.

[43] Hwang, M.Y. and X.D. Huang, "Dynamically Configurable Acoustic Modelings for Speech Recognition," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1998, Seattle, WA.

[44] Jain, A., J. Mao, and K.M. Mohiuddin, "Artifical Neural Networks: A Tutorial," *Compter*, 1996, **29**(3), pp. 31-44.

[45] Jayant, N.S. and P. Noll, *Digital Coding of Waveforms*, 1984, Englewood Cliffs, NJ, Prentice-Hall.

[46] Jiang, L., H.W. Hon, and X.D. Huang, "(don't use) Improvements on a Trainable Letter-to-Sound Converter," *Eurospeech'97*, 1997, Rhodes, Greece.

[47] Juang, B.H., W. Chou, and C.H. Lee, "Statistical and Discriminative Methods for Speech Recognition" in *Automatic Speech and Speaker Recognition - Advanced Topics*, C.H. Lee, F.K. Soong, and K.K. Paliwal, eds. 1996, Boston, pp. 109-132, Kluwer Academic Publishers.

[48] Juang, B.H. and S. Katagiri, "Discriminative Learning for Minimum Error Classification," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1992, **SP-40**(12), pp. 3043-3054.

[49] Kanal, L.N. and B. Chandrasekaran, "On Dimensionality and Sample Size in Statistical Pattern Classification," *Proc. of NEC*, 1968, **24**, pp. 2-7.

[50] Kanal, L.N. and N.C. Randall, "Recognition System Design by Statistical Analysis," *ACM Proc. of 19th National Conf.*, 1964 pp. D2.5-1-D2.5-10.

[51] Kohonen, T., *Learning Vector Quantization for Pattern Recognition*, 1986, Helsinki University of Technology, Finland.

[52] Kuhn, R. and R.D. Mori, "The Application of Semantic Classification Trees to Natural Language Understanding," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 1995(7), pp. 449-460.

[53] Lachenbruch, P.A. and M.R. Mickey, "Estimation Of Error Rates In Discriminant Analysis," *Technometrics*, 1968, **10**, pp. 1-11.

[54] Laplace, P.S., *Theorie Analytique des Probabilities*, 1812, Paris, Courcier.

[55] Lee, D.S., S.N. Srihari, and R. Gaborski, "Bayesian and Neural Network Pattern Recognition: A Theoretical Connection and Empirical Results with Handwriting Characters" in *Artificial Neural Network and Statistical Pattern Recognition: Old and New Connections*, I.K. Sethi and A.K. Jain, eds. 1991, Amsterdam, North-Holland.

[56] Linde, Y., A. Buzo, and R.M. Gray, "An Algorithm for Vector Quantizer Design," *IEEE Trans. on Communication*, 1980, **COM-28**(1), pp. 84-95.

[57] Lippmann, R.P., "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, 1987, pp. 4--22.

[58] Lippmann, R.P., "Review of Neural Nets for Speech Recognition," *Neural Computation*, 1989, **1**, pp. 1-38.

[59] Lloyd, S.P., "Least Squares Quantization in PCM," *IEEE Trans. on Information Theory*, 1982, **IT-2**, pp. 129-137.

[60] Lucassen, J.M. and R.L. Mercer, "An Information-Theoretic Approach to the Automatic Determination of Phonemic Baseforms," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1984, San Diego, California pp. 42.5.1-42.5.4.

[61] Makhoul, J., S. Roucos, and H. Gish, "Vector Quantization in Speech Coding," *Proc. of the IEEE*, 1985, **73**(11), pp. 1551-1588.

[62] Martin, J.K., "An Exact Probability Metric For Decision Tree Splitting And Stopping," *Articial Intelligence and Statistics*, 1995, **5**, pp. 379-385.

[63] McClelland, J., "The Programmable Blackboard Model" in *Parallel Distributed Processing - Explorations in the Microstructure of Cognition, Volume II: Psychological and Biological Models* 1986, Cambridge, MA, MIT Press.

[64] McCulloch, W.S. and W. Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, 1943.

[65] McLachlan, G. and T. Krishnan, *The EM Algorithm and Extensions*, 1996, New York, NY, Wiley Interscience.

[66] McNemar, E.L., "Note on the Sampling Error of The Difference Between Correlated Proportions or Percentages," *Psychometrika*, 1947, **12**, pp. 153-157.

[67] Meisel, W.S., *et al.*, "The SSI Large-Vocabulary Speaker-Independent Continuous Speech Recognition System" 1991, pp. 337-340.

[68] Minsky, M. and S. Papert, *Perceptrons*, 1969, Cambridge, MA, MIT Press.

[69] Mitchell, T., *Machine Learning*, McGraw-Hill Series in Computer Science, 1997, McGraw-Hill.

[70] Morgan, J.N. and J.A. Sonquist, "Problems in the Analysis of Survey Data and a Proposal," *Journal of American Statistics Association*, 1962, **58**, pp. 415-434.

[71] Nadas, A., "A Decision-Theoretic Formulation of a Training Problem in Speech Recognition and a Comparison of Training by Unconditional Versus Conditional Maximum Likelihood," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1983, **4**, pp. 814-817.

[72] Newton, I., *Philosophiae Naturalis Principlea Mathematics*, 1687, London, Royal Society Press.

[73] Neyman, J. and E.S. Pearson, "On the Problem of the Most Efficient Tests of Statistical Hypotheses," *Philosophical Trans. of Royal Society*, 1928, **231**, pp. 289-337.

[74] Ostendorf, M. and N. Velleux, "A Hierarchical Stochastic Model for Automatic Prediction of Prosodic Boundary Location," *Computational Linguistics*, 1995, **20**(1), pp. 27--54.

[75] Paliwal, K. and W.B. Kleijn, "Quantization of LPC Parameters" in *Speech Coding and Synthesis*, W.B. Kleijn and K.K. Paliwal, eds. 1995, Amsterdam, pp. 433-466, Elsevier.

[76] Paul, D.B., "An 800 bps Adaptive Vector Quantization in Speech Coding," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1983 pp. 73-76.

[77] Paul, D.B., "The Lincoln Tied-Mixture HMM Continuous Speech Recognizer" in *Morgan Kaufmann Publishers* 1990, San Mateo, CA, pp. 332-336.

[78] Payne, H.J. and W.S. Meisel, "An Algorithm for Constructing Optimal Binary Decision Trees," *IEEE Trans. on Computers*, 1977, **C-26**(September), pp. 905-916.

[79] Pearson, K., "Contributions to The Mathematical Theorem of Evolution," *Philosophical Trans. of Royal Society*, 1894, **158A**, pp. 71-110.

[80] Pitrelli, J. and V. Zue, "A Hierarchical Model for Phoneme Duration in American English" in *Proc. of Eurospeech* 1989.

[81] Press, W.H., *et al.*, *Numerical Receipes in C*, 1988, Cambridge, UK, Cambridge University Press.

[82] Quinlan, J.R., "Introduction of Decision Trees" in *Machine Learning : An Artifical Intelligence Approach*, R. Michalski, J. Carbonell, and T. Mitchell, eds. 1986, Boston, M.A., pp. 1-86, Kluwer Academic Publishers.

[83] Quinlan, J.R., "Generating Production Rules from Decision Trees," *Int. Joint Conf. on Artifical Intelligence*, 1987 pp. 304-307.

[84] Quinlan, J.R., *C4.5: Programs for machine learning*, 1993, San Francisco, Morgan Kaufmann.

[85] Rabiner, L. and B.H. Juang, "Speech Recognition System Design and Implementation Issues" in *Fundamental of Speech Recognition*, L. Rabiner and B.H. Juang, eds. 1993, Englewood Cliffs, NJ, pp. 242-340, Prentice Hall.

[86] Reddy, D.R., "Speech Recognition by Machine: A Review," *IEEE Proc.*, 1976, **64**(4), pp. 502-531.

[87] Rosenblatt, F., "The Perceptron --- A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review*, 1958, **65**, pp. 386-408.

[88] Rummelhart, D.E., G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation" in *Parallel Distributed Processing*, D.E. Rumelhart and J.L. McClelland, eds. 1986, Cambridge, MA, pp. 318-362, MIT Press.

[89] Rummelhart, D.E., G.E. Hinton, and R.J. Williams, "Learning Representations by Back-Propagating Errors," *Nature*, 1986, **323**, pp. 533-536.

[90] Rummelhart, D.E. and J.L. McClelland, *Parallel Distributed Processing - Explorations in the Microstructure of Cognition, Volume I : Foundations*, 1986, Cambridge, MA, MIT Press.

[91] Rummelhart, D.E. and J.L. McClelland, *Parallel Distributed Processing - Explorations in the Microstructure of Cognition, Volume II: Psychological and Biological Models*, 1986, Cambridge, MA, MIT Press.

[92] Schalkoff, R.J., *Digital Image Processing and Computer Vision*, 1989, New York, NY, John Wiley & Sons.

[93] Sejnowski, T.J. and C.R. Rosenberg, "Parallel Networks that Learn to Pronounce English Text," *Complex Systems*, 1987, **1**, pp. 145-168.

[94] Sejnowski, T.J., *et al.*, "Combining Visual And Acoustic Speech Signals with a Neural Network Improve Intelligibility" in *Advances in Neural Information Processing Systems* 1990, San Mateo, CA, pp. 232-239, Morgan Kaufmann.

[95] Sparkes, J.J., "Pattern Recognition and a Model of the Brain," *Int. Journal of Man-Machine Studies*, 1969, **1**(3), pp. 263-278.

[96] Tappert, C., C.Y. Suen, and T. Wakahara, "The State of the Art in On-Line Handwriting Recognition," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 1990, **12**(8), pp. 787-808.

[97] Tesauro, G. and T.J. Sejnowski, "A Neural Network That Learns to Play Backgammon," *Neural Information Processing Systems, American Institute of Physics*, 1988, pp. 794-803.

[98] White, H., "Learning in Artificial Neural Networks: A Statistical Perspective," *Neural Computation*, 1989, **1**(4), pp. 425--464.

[99] Winston, P.H., *Artificial Intelligence*, 1984, Reading, MA, Addison-Wesley.

[100] Wu, C.F.J., "On The Convergence Properties of the EM Algorithm," *The Annals of Statistics*, 1983, **11**(1), pp. 95-103.

[101] Young, J.Z., *Programmes of the Brain*, 1975, Oxford, England, Oxford University Press.